

UNIVERSITY OF CALIFORNIA

Los Angeles

Stock Price Prediction using Adaptive Time Series Forecasting and Machine  
Learning Algorithms

A thesis submitted in partial satisfaction

of the requirements for the degree Master of Applied Statistics

by

Lumeng Chen

2020

© Copyright by

Lumeng Chen

2020

## ABSTRACT OF THE THESIS

### Stock Price Prediction using Adaptive Time Series Forecasting and Machine Learning Algorithms

by

Lumeng Chen

Master of Applied Statistics

University of California, Los Angeles, 2020

Professor Yingnian Wu, Chair

In this thesis, ARIMA model, Long Short Term Memory (LSTM) model and Extreme Gradient Boosting (XGBoost) models were developed to predict daily adjusted close price of selected stocks from January 3, 2017 to April 24, 2020. Daily stock price data includes columns of open, close, adjusted close, high, low and volume. In ARIMA and LSTM models, the only features we used as model inputs were previous N days' stock prices. Prediction on day N+1 was calculated based on previous N values. RMSE and MAPE were calculated from this rolling forecast and the actual price in the test dataset. Optimal parameters were selected to be the setting that yielded the lowest RMSE score. Residuals diagnostic was performed to check model assumption for the final ARIMA model. In XGBoost model, feature engineering was used to create two additional features from open, close, high and low price. Same with LSTM model, previous N days features were used as

features in day  $N+1$  for prediction. In both LSTM and XGBoost models, training dataset was scaled for model fitting. Features and output from cross-validation and test dataset were scaled too based on previous  $N$  days' values. The prediction results were then reverted back to original scale before calculation of RMSE and MAPE scores.

In conclusion, looking at the prediction versus actual stock price plot for each stock and their RMSE and MAPE scores, all three models produced good forecast of next day's stock price. However, during the time with great volatility, the lag between forecast value and actual value is more noticeable. In our models, historical  $N$  days stock price on its own could provide a relatively accurate prediction on  $N+1$  day's stock price. In XGBoost model particularly, we found out that  $N=2$  provided better RMSE and MAPE(%) results than other larger values of  $N$  (previous  $N$  days). As  $N$  gets larger, prediction accuracy got lower in XGBoost. In XGBoost feature importance analysis, the most important factor to today's stock price is its price yesterday. Although the final ARIMA model achieved the lowest RMSE score, grid search for one-step ARIMA forecast model parameters took the longest computing time, while XGBoost model with the second lowest RMSE score required the least time for parameter tuning and forecast calculation.

The thesis of Lumeng Chen is approved.

Frederic R. Paik Schoenberg

Nicolas Christou

Yingnian Wu, Committee Chair

University of California, Los Angeles

2020

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1 Introduction and Outline.....	1
1.2 Data Description .....	2
1.3 Comparison Metrics – RMSE and MAPE.....	4
<b>2. Descriptive Time Series Analysis.....</b>	<b>5</b>
2.1 Descriptive Data Analysis .....	5
<b>3. ARIMA Model.....</b>	<b>7</b>
3.1 Methodology.....	7
3.2 ARIMA model fitting of VTI stock price.....	9
3.3 Tuning p, d, q.....	10
3.4 ARMA(8,0,1) Model Diagnostic.....	13
3.5 Comparison with SARIMA model .....	15
3.6 Ljung Box Test .....	17
3.7 Model Fitting for MSFT, GOOG and AMZN .....	18
<b>4. Long Short Term Memory Model .....</b>	<b>18</b>
4.1 Methodology.....	18
4.2 Model Fitting of VTI stock price.....	21
4.3 Parameter Tuning .....	21

4.4 Final Model and Prediction results .....	25
4.5 Model fitting for MSFT, AMZN and GOOG .....	26
<b>5. Extreme Gradient Boosting Model.....</b>	<b>26</b>
5.1 Methodology.....	26
5.2 Model Development of VTI stock price .....	29
5.3 Parameter Tuning .....	32
5.4 Final Model and Prediction results .....	34
5.5 Model Fitting for MSFT, AMZN and GOOG .....	36
<b>6. Conclusion and Recommendation .....</b>	<b>36</b>
6.1 Conclusion .....	36
6.2 Recommendation .....	37
<b>Appendix A Figures and Tables .....</b>	<b>39</b>
<b>References .....</b>	<b>51</b>

## List of Tables

Table 1 VTI stock price data.....	3
Table 2 Training and test dataset split .....	3
Table 3 Data summary of VTI, AMZN, MSFT and GOOG .....	6
Table 4 Ljung-Box Test.....	18
Table 5 ARIMA model results for MSFT, GOOG and AMZN .....	18
Table 6 Tuning optimizer.....	24
Table 7 LSTM parameters after tuning.....	25
Table 8 LSTM model results for MSFT, GOOG and AMZN .....	26
Table 9 RMSE and MAPE (%) for N .....	30
Table 10 VTI training data.....	31
Table 11 XGBoost parameters after tuning .....	35
Table 12 XGBoost model results for MSFT, GOOG and AMZN.....	36
Table 13 Model Comparison .....	36
Table 14 ARIMA model Grid search RMSE and MAPE results .....	41
Table 15 SARIMA(p, d, q) x (P, D, Q, S) grid search output .....	43
Table 16 Tuning N in LSTM; Tuning epochs and batch_size in LSTM; Tuning LSTM_units and dropout_prob .....	44
Table 17 Tuning n_estimators and max_depth; Tuning learning_rate and min_child_weight ....	46
Table 18 Tuning subsample and gamma parameter; Tuning colsample_bytree and colsample_bylevel.....	47



## List of Figures

Figure 1 Correlation plot.....	6
Figure 2 Stock price trend charts .....	7
Figure 3 VTI adjusted close price autocorrelation lag plot and distribution plot .....	9
Figure 4 VTI stock price forecast using ARIMA .....	13
Figure 5 Residuals plot .....	14
Figure 6 ACF and PACF of residuals .....	14
Figure 7 VTI stock price forecast using SARIMA(1, 1, 1) x (0, 1, 1, 30).....	16
Figure 8 Standard RNN repeating module with single layer [5] .....	19
Figure 9 LSTM repeating module with interacting layers [5] .....	19
Figure 10 LSTM model flowchart .....	20
Figure 11 RMSE/MAPE(%) with N .....	22
Figure 12 Tuning epochs and batch_size.....	23
Figure 13 Tuning LSTM units and dropout probability .....	24
Figure 14 VTI stock price prediction on LSTM test dataset .....	25
Figure 15 VTI price without scaling and training series with scaling .....	31
Figure 16 Tuning n_estimators and max_depth.....	32
Figure 17 Tuning subsample and gamma parameters.....	33
Figure 18 Tuning colsample_bytree and colsample_bylevel.....	34
Figure 19 VTI stock price prediction on XGBoost test dataset .....	35
Figure 20 Time Series Decomposition of VTI train data .....	39
Figure 21 MSFT stock price forecast using ARIMA.....	39
Figure 22 GOOG stock price forecast using ARIMA .....	40

Figure 23 AMZN stock price forecast using ARIMA .....	40
Figure 24 MSFT stock price forecast using LSTM .....	47
Figure 25 AMZN stock price forecast using LSTM.....	48
Figure 26 GOOG stock price forecast using LSTM .....	48
Figure 27 MSFT stock price forecast using XGBoost.....	49
Figure 28 GOOG stock price forecast using XGBoost.....	49
Figure 29 AMZN stock price forecast using XGBoost .....	50

# 1. Introduction

## 1.1 Introduction and Outline

Machine learning algorithms are known to be very effective in prediction problems. One of its application is predicting time series. In this paper, we applied a few adaptive models, XGBoost, LSTM and ARIMA models to predict stock price of Vanguard Total Stock Market ETF (ticker: VTI), Amazon (ticker: AMZN), Microsoft (ticker: MSFT) and Google (ticker: GOOG). Predicting stock price with 100% accuracy is almost impossible in reality as stock price is heavily influenced by unquantifiable factors, such as market irrational behavior and news. In this thesis, forecast model fitting and parameter tuning for VTI adjusted close price is discussed in detail as an example. For other stocks, we applied the same model development procedure as illustrated in VTI and compared their prediction results to see if model performance is consistent across different stocks. Unlike other models where trading volume, opening price and other technical indicators are fed into the model, we built our ARIMA and LSTM models only use adjusted close price in previous  $N$  days as input to predict price on day  $N+1$ . This is because although stock price might be affected by other factors, it is largely dependent on historical stock price. In XGBoost model, difference between high and low price; difference between open and close price and volume were added as features to the model through feature engineering. All features from previous  $N$  days were added as features to predict stock price on  $N+1$  day too.

In chapter 1, we briefly introduced selected stock portfolio and the performance metrics chosen for models' forecast performance comparison. In chapter 2, data summary of stock data of VTI, MSFT, GOOG and AMZN was reviewed. We found out that they are highly correlated but with significantly different volatility (standard deviation). In the following chapters, we investigated if model prediction performance for each method can remain consistent across various stocks. In

chapter 3, ARIMA rolling forecast model was built on the training dataset of VTI stock price time series and grid search for optimal parameter settings was performed. RMSE score was calculated to find the optimal parameter setting. Residuals diagnostic was conducted to check model assumption. Seasonal ARIMA (SARIMA) one-step forecast model was also examined to see if it has better prediction accuracy based on RMSE score. Grid search of SARIMA model parameters was performed too and the optimal parameter setting was selected based on AIC. With the same process, ARIMA models have been developed for the other three stocks. RMSE and MAPE (%) were calculated for models' comparison. In chapter 4 and chapter 5, we went through LSTM and XGBoost model methodology respectively. Original parameters were used to develop model on training dataset of VTI stock price. Model parameters were tuned on development dataset and model prediction performance was assessed on test dataset. The same process was performed for the other stocks data. In chapter 6, prediction metrics were compared and discussion has been made for LSTM, ARIMA and XGBoost models. Future potential research topic for stock price forecasting was mentioned.

## 1.2 Data Description

Four stocks' data, Amazon (ticker: AMZN), Microsoft (ticker: MSFT), Vanguard Total Stock Market ETF (ticker: VTI) and Google (ticker: GOOG), are collected from Yahoo! Finance. Each dataset has attributes of open, high, low, closing, adjusted closing price and trading volume as shown below.

<b>date</b>	<b>open</b>	<b>high</b>	<b>low</b>	<b>close</b>	<b>adj_close</b>	<b>volume</b>
2017-01-03	116.150002	116.550003	115.489998	116.199997	109.298668	2731600
2017-01-04	116.470001	117.190002	116.419998	117.089996	110.135796	3228700
2017-01-05	116.949997	117.089996	116.400002	116.860001	109.919472	2604000
2017-01-06	117.019997	117.519997	116.639999	117.230003	110.267487	2317600
2017-01-09	117.099998	117.139999	116.739998	116.779999	109.844208	2461400

Table 1 VTI stock price data

For all stock indices, daily adjusted closing price is used as response variable in each model. Adjusted closing price is obtained by factoring in any factor, for example, corporate actions such as stock splits, dividends or distributions, which might affect the stock price to the closing price after the market closes. It is considered to be the true price of a stock and is often used when examining historical returns or performing a detailed analysis historical return. For model prediction, we are only interested in forecasting adjusted closing price for simplicity and consistency. It is understood that model comparison results should hold if we chose other types of price as dependent variable for the same index. When building Extreme Gradient Boosting (XGBoost) model, all other data points, open, close, high, low prices and trading volume were used too as input for the model. For other models, only daily adjusted closing price from previous days with different lags were fed into the model to determine the optimal parameter setting.

LSTM and XGBoost			
Period	Start	End	Length
Training	Jan-17	Dec-18	501
Cross-validation	Jan-19	Aug-19	166
Test	Sep-19	Apr-20	166
ARIMA			
Period	Start	End	Length
Training	Jan-17	Aug-19	666
Test	Sep-19	Apr-20	167

Table 2 Training and test dataset split

In terms of time period selection, January 3, 2017 to April 24, 2020 is selected for training and test datasets for all models with the split shown in table 2. In total, there are 833 observations in the dataset. For Long Short Term Memory model (LSTM) and XGBoost models, split for training, cross-validation (development) and test dataset are 60%, 20% and 20% respectively. Cross-validation dataset is created to tune models' hyperparameters. In ARIMA model development, 80% of the data is used for training dataset and 20% is for test dataset. Number of observations in each dataset is listed in table 2 above. Models were trained on the training set and model performance were reported on the test dataset.

### 1.3 Comparison Metrics – RMSE and MAPE

Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) are statistical measure of how accurate a forecast model is. RMSE is used to measure difference between actual values and forecasted in the test dataset. It represents the square root of the second sample moment of the differences between predicted values and observed values. RMSE aggregates the magnitudes of the errors in predictions across various time periods into a single measure of predictive power by taking the average of the error by number of fitted points. RMSE is a measure of accuracy, to compare forecasting errors of different models for the same dataset as it is scale-dependent. RMSE is calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{t=1}^n (F_t - A_t)^2}{n}}$$

MAPE measures prediction accuracy of a forecast model. It is calculated by summing the ratio of difference between actual value and forecasted value over actual value, then take the average of

this sum by the number of fitted points  $n$ . It usually expresses the accuracy as a ratio defined by formula:

$$\text{MAPE} = \frac{1}{n} \sum_{t=1}^n \left| \frac{F_t - A_t}{A_t} \right|, \quad [18]$$

where  $A_t$  is the actual value and  $F_t$  is the forecast value. Sometimes, because MAPE takes division of error by the actual value individually, it has the risk of being too skewed. A big error with a low actual value has a huge impact on MAPE. Due to this, model's parameter tuning was based on RMSE score, as optimizing MAPE may result in an inaccurate forecast which is most likely to undershoot the actual value.

## 2. Descriptive Time Series Analysis

### 2.1 Descriptive Data Analysis

Below figure 2 demonstrated trends of selected stocks during selected time frame. MSFT, GOOG and AMZN has similar trend with VTI, however, AMZN and GOOG has much greater price range than MSFT and VTI. AMZN's adjusted closing price ranged from \$1,000 from the beginning of 2017 to currently around \$2,500, while VTI and MSFT's price range are historically under \$200. Additionally, as can be seen from correlation plot (figure 1), all four stocks are highly correlated with each other with correlation coefficient higher than 0.80. This is because MSFT, GOOG and AMZN share the similar growth trends as they are all technology companies with similar characteristics. In addition, MSFT, AMZN and GOOG make up 4.77%, 3.26% and 1.35% respectively of the VTI portfolio<sup>1</sup>. It is reasonable for them to have such high correlation. However, as can be seen from table 3, GOOG and MSFT have much higher volatility (standard deviation)

than MSFT and VTI while VTI has the lowest volatility. Whether or not our forecasting models share the similar forecasting performance for all stocks are investigated in later chapters.

	VTI	AMZN	MSFT	GOOG
<b>count</b>	833	833	833	833
<b>mean</b>	135.48	1,516.50	105.15	1,098.57
<b>std</b>	14.22	400.37	32.34	154.99
<b>min</b>	109.30	753.67	58.87	786.14
<b>25%</b>	124.49	1,100.95	78.82	992.81
<b>50%</b>	135.67	1,643.24	102.77	1,102.23
<b>75%</b>	144.96	1,823.29	133.16	1,194.43
<b>max</b>	171.32	2,410.22	188.19	1,526.69

Table 3 Data summary of VTI, AMZN, MSFT and GOOG

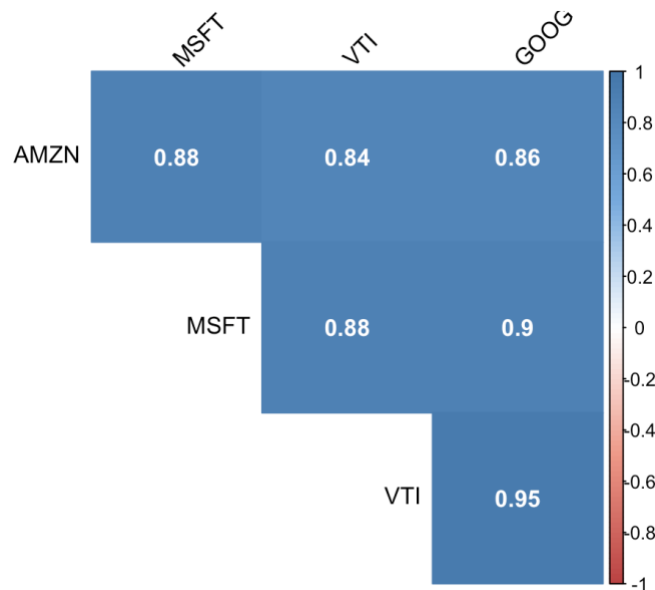


Figure 1 Correlation plot



### Time Series with Rangeslider - VTI, MSFT, GOOG and AMZN



Figure 2 Stock price trend charts

## 3. ARIMA Model

### 3.1 Methodology

ARIMA stands for Auto-Regressive Integrated Moving Averages.  $ARIMA(p, d, q)$  is a generalization of an autoregressive moving average ( $ARMA(p, q)$ ) model. ARIMA models are applied in some cases where data show evidence of non-stationarity. The AR term  $p$  of ARIMA

indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior) values. For example, if  $p$  is 3, the predictors for  $y_t$  will be  $y_{t-1}, \dots, y_{t-3}$ . The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. For example, if  $q$  is 5, the predictors for  $y_t$  will include  $\varepsilon_{t-1}, \dots, \varepsilon_{t-5}$ . The I (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once) [1]. Generalized ARMA( $p, q$ ) process is defined as below [1]:

$$E(\varepsilon_t, \varepsilon_s) = 0, \text{ for } t \neq s$$

$$y_t = \mu + a_t y_{t-1} + \dots + a_k y_{t-p} + \varepsilon_t + b_1 \varepsilon_{t-1} + \dots + b_q \varepsilon_{t-q} \quad [1]$$

Where  $a_i$  are the parameters of the autoregressive part of the model and  $b_i$  are the parameters of the moving average part and  $\varepsilon_i$  are the error terms.

As a simplified notation, this is often expressed in terms of lag-polynomials as

$$\Phi(L)y_t = \psi(L) \varepsilon_t \quad [1]$$

Where

$$\Phi(L) = 1 - a_1 L^1 - a_2 L^2 - \dots - a_k L^p$$

$$\psi(L) = 1 + b_1 L^1 + b_2 L^2 + \dots + b_q L^q \quad [1]$$

$L$  is the lag or shift operator,  $L^i x_t = x_{t-i}$ ,  $L^0 = 1$ . [1]

Seasonal Arima (SARIMA) model was also examined to see if it can provide better prediction for stock price. SARIMA( $p, d, q$ )( $P, D, Q, S$ ) includes non-seasonal orders:  $p$  for autoregressive order;  $d$  for differencing order and  $q$  for moving average order. ( $P, D, Q, S$ ) is seasonal orders:  $P$  for seasonal autoregressive order;  $D$  for seasonal differencing order;  $Q$  for seasonal moving average

order and  $S$  is number of time steps per cycle [17], in other words, the specified periodicity of time series.

### 3.2 ARIMA model fitting of VTI stock price

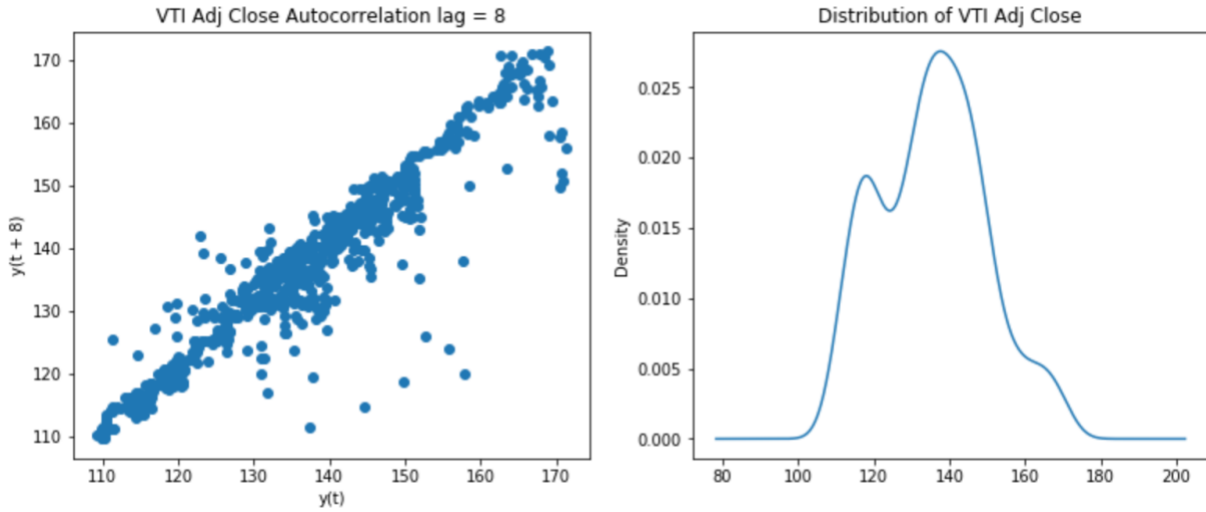


Figure 3 VTI adjusted close price autocorrelation lag plot and distribution plot

As can be observed from figure 3, autocorrelation lag plot of VTI adjusted close price closely follows a straight line. Its distribution looks normal. Both plots suggest that ARIMA would be a good model for the data.

VTI adjusted close price series was split into 80% training data and 20% test data. One-step forecast was performed over the test dataset. We used training dataset to fit the model and generate a prediction for each day in the test dataset. A rolling forecast was performed given the dependence on observations in prior time steps. An ARIMA model was recreated after each new observation is received to generate the rolling forecast. All observations are manually tracked and stored in a list called history that is seeded with the training data and to which new observations are appended

each iteration. Below is the code we used to generate rolling forecast from ARIMA model and calculate MAPE and RMSE score on test dataset [19]:

```
train_ar = train_data['adj_close'].values
test_ar = test_data['adj_close'].values

history = [x for x in train_ar]
print(type(history))
predictions = list()
for t in range(len(test_ar)):
    model = ARIMA(history, order=(8,0,1))
    model_fit = model.fit(dispatch=0)
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test_ar[t]
    history.append(obs)
error = math.sqrt(mean_squared_error(test_ar, predictions))
print('Residual Mean Squared Error: %.3f % error)
error2 = mape_fun(test_ar, predictions)*100
print('mean absolute percentage error: %.3f % error2)
```

### 3.3 Tuning p, d, q

Grid search for optimal (p, d, q) setting was performed on training dataset. RMSE and MAPE scores were calculated from prediction on the test dataset and the actual value in test dataset. The

(p, d, q) combination yielded lowest RMSE was selected as final ARIMA model using code [20]  
below:

```
# evaluate an ARIMA model for a given order (p,d,q)

def evaluate_arma_model(X, arma_order):

    train_size = int(len(X) * 0.8) # prepare training dataset

    train, test = X[0:train_size], X[train_size:]

    history = [x for x in train]

    predictions = list()

    for t in range(len(test)):

        model = ARIMA(history, order=arma_order)

        model_fit = model.fit(dispatch=0)

        yhat = model_fit.forecast()[0]

        predictions.append(yhat)

        history.append(test[t])

    rmse = math.sqrt(mean_squared_error(test, predictions)) # calculate out of sample error

    return rmse

# evaluate combinations of p, d and q values for an ARIMA model

def evaluate_models(dataset, p_values, d_values, q_values):

    dataset = dataset.astype('float32')

    best_score, best_cfg = float("inf"), None

    for p in p_values:

        for d in d_values:

            for q in q_values:

                order = (p,d,q)

                try:
```

```

rmse = evaluate_arima_model(dataset, order)

if rmse < best_score:

    best_score, best_cfg = rmse, order

    print('ARIMA%s RMSE=%.3f' % (order,rmse))

except:

    continue

print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))

# evaluate parameters

p_values = [0, 1, 2, 4, 6, 8]

d_values = range(0, 3)

q_values = range(0, 3)

evaluate_models(df['adj_close'].values, p_values, d_values, q_values)

```

*Residual Mean Squared Error: 2.985*

*Mean Absolute Percentage Error: 9.675*

As can be seen from logic above, p value was selected from 0, 1, 2, 4, 6, 8 and ranges for d and q values were from 0 to 3. Grid search results are provided in table 14 in Appendix A. In the output results, RMSE for ARIMA models ranged from 2.985 to 22.457 and MAPE ranged from 9.513% to 12.457%. ARIMA (8, 0, 1) produced the lowest RMSE of 2.985 with 9.675% MAPE, while the lowest MAPE was produced by ARIMA (0, 0, 1) with 11.978 RMSE. In comparison, one-step forecast model, ARIMA(8, 0, 1), has the lowest value in RMSE and relatively low MAPE metrics. Therefore, ARIMA(8, 0, 1) was selected as forecast model for VTI stock price and the next step is to check residuals diagnostics.

As can be seen from figure 4, one-step forecast using ARIMA model provided accurate predictions on the next day's adjusted close price despite the great volatility stock market experienced in the

beginning of 2020. MAPE score of approximately 9.675% indicates the model is approximately 90.325% accurate in predicting test set observations.

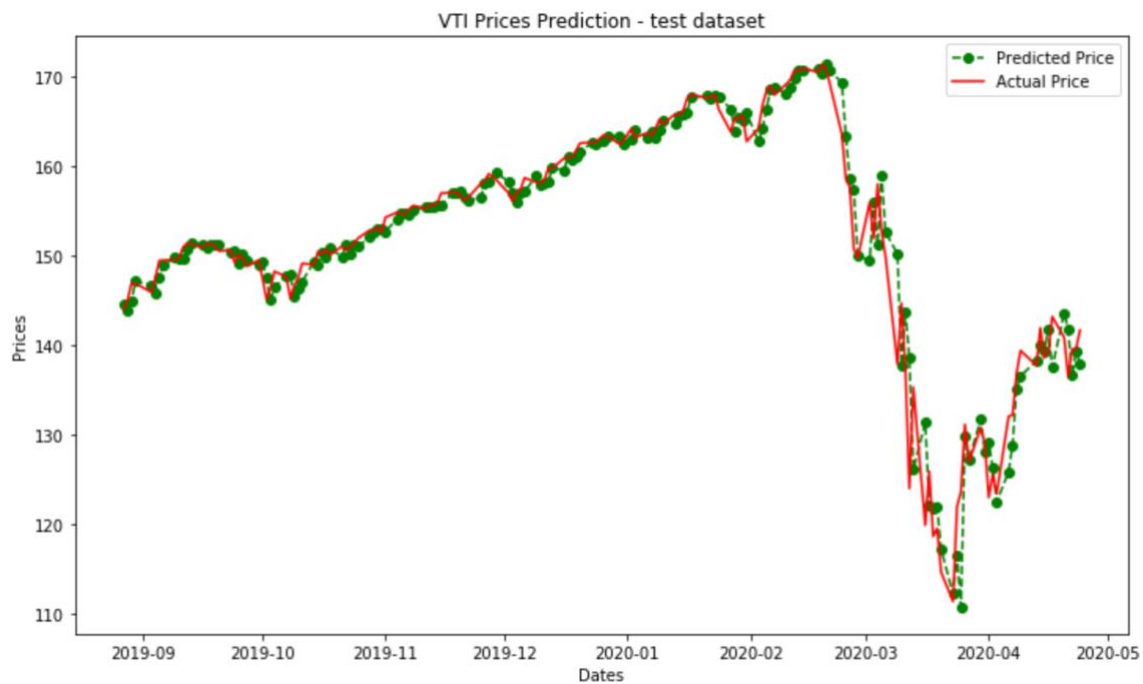


Figure 4 VTI stock price forecast using ARIMA

### 3.4 ARMA(8,0,1) Model Diagnostic

As can be observed from figure 5 below, the residuals versus order plot (left) indicates that the residuals did not violate the assumption of constant location and scale. Residuals appeared to be random with most of points scattered in the range of  $(-1, 1)$ . From residuals versus fitted plot on the right, we can see that the most of residual points fall randomly on both sides of 0, with no recognizable patterns in the points. The fitted red line is also a relatively flat line. Based on the plot, it is reasonable to conclude that residuals are unbiased and have a constant variance. Therefore, the forecast is accurate.

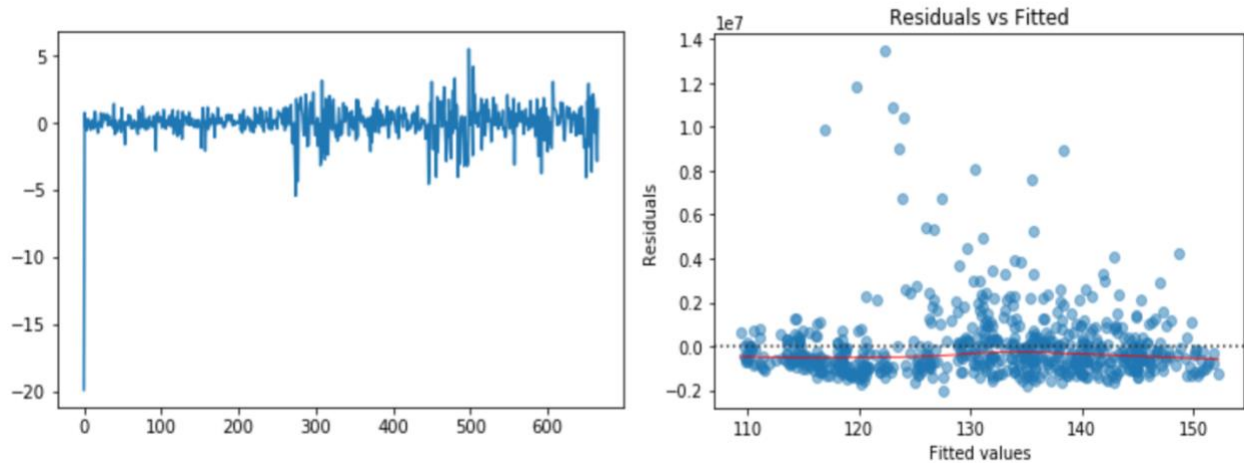


Figure 5 Residuals plot

Autocorrelation and partial autocorrelation plots for residuals are also reviewed below. The autocorrelation function is a measure of the correlation between the observations of a time series that are separated by  $k$  time units. The partial autocorrelation function is a measure of the correlation between the observations of a time series that are separated by  $k$  time units ( $y_t$  and  $y_{t-k}$ ), after adjusting for the presence of all the other terms of shorter lag ( $y_{t-1}, y_{t-2}, \dots, y_{t-k-1}$ ) [16]. From below autocorrelation and partial correlation plot, since there are no significant correlations present (correlation lags exceed the blue shaded confidence interval), we can conclude that model meets assumption and the residuals are independent. Ljung-box test will be used later to check residuals' randomness again later.

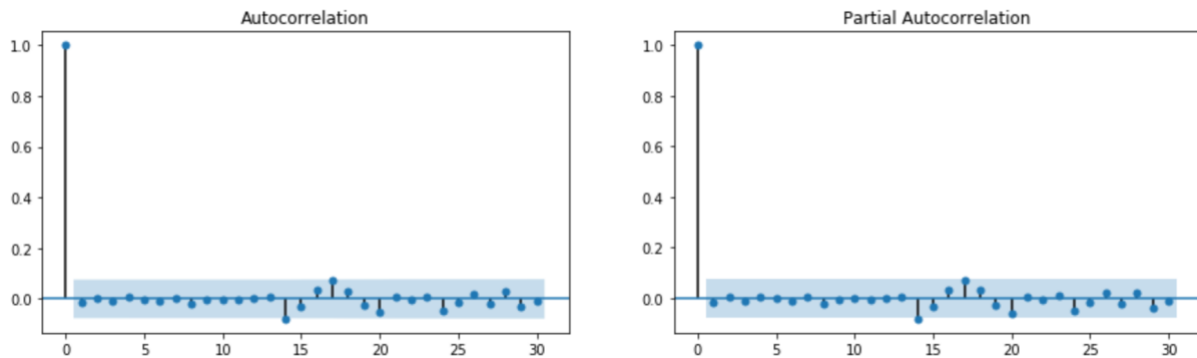


Figure 6 ACF and PACF of residuals



### 3.5 Comparison with SARIMA model

Since seasonality was not considered in above ARIMA model, here we checked if including seasonal parameter could fit the training data better, in other words, check if the series has seasonality. SARIMA(p, d, q) x (P, D, Q, S) model which models seasonal effect in a multiplicative way is used. We used a “grid search” to iteratively explore different combinations of parameters. For each combination of parameters, we fitted a new seasonal ARIMA model with the SARIMAX() function from the statsmodels module [3] and assess its overall quality. P and q were set to iterate between 0 and 2 as shown below [14]:

```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(train_ar,
                                             order=param,
                                             seasonal_order=param_seasonal,
                                             enforce_stationarity=False,
                                             enforce_invertibility=False)

            results = mod.fit()

            print('ARIMA {} x {} 30 - AIC: {}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

with example output as follows:

*ARIMA(0, 0, 1)x(0, 1, 0, 30)30 – AIC: 3271.08542290716*

*ARIMA(0, 0, 1)x(0, 1, 1, 30)30 - AIC: 3128.19644907746*

Different combinations of SARIMA(p, d, q) x (P, D, Q, S) were produced. Within those models, Akaike information criterion (AIC) was used to select the optimal SARIMA model. AIC estimates the relative amount of information lost by a given model: the less information a model loses, the higher the quality of that model. All generated seasonal ARIMA model is provided in table 15 in Appendix. SARIMA model with parameter (1, 1, 1) x (0, 1, 1, 30) has the lowest AIC and was selected as optimal SARIMA model.

One-step-ahead prediction was calculated and RMSE for this model is 8.538. From below prediction plot under SARIMA model we can see that there are a few significant outliers in predictions. Also, seasonality in the predictions of test dataset are exaggerated since green dotted line fluctuates more notably than red line of actual values. This may give rise to the higher RMSE score. As RMSE of one-step forecast SARIMA model was much greater than RMSE of rolling forecast ARIMA(8, 0, 1) model, ARIMA(8, 0, 1) was selected as optimal model for VTI stock price forecast.

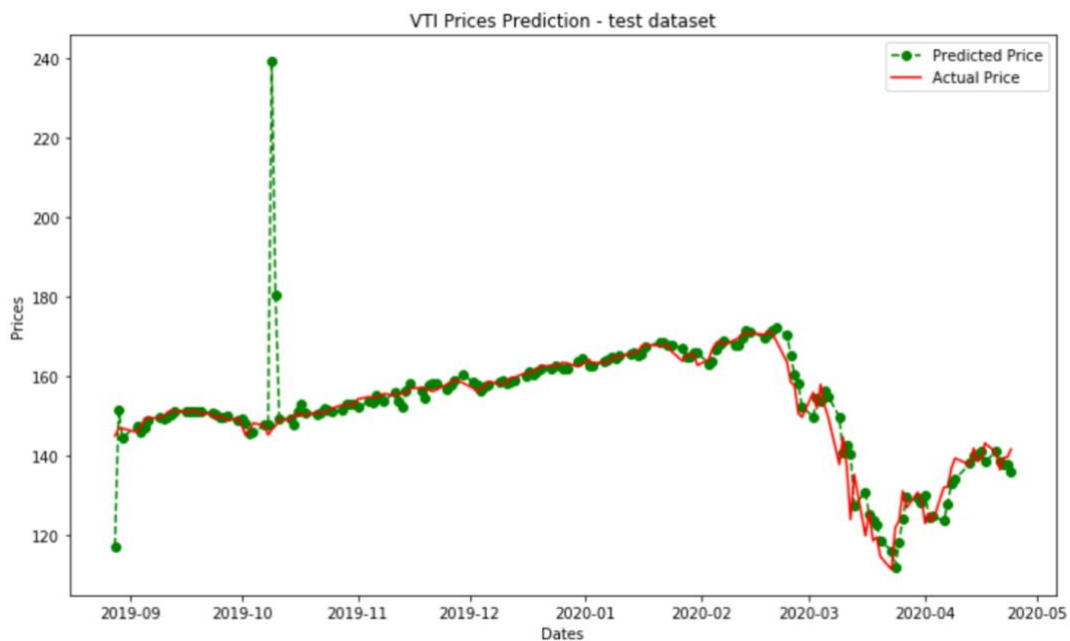


Figure 7 VTI stock price forecast using SARIMA(1, 1, 1) x (0, 1, 1, 30)

### 3.6 Ljung Box Test

The Ljung–Box test (named for Greta M. Ljung and George E. P. Box) is a type of statistical test of whether any of a group of autocorrelations of a time series are different from zero. Instead of testing randomness at each distinct lag, it tests the "overall" randomness based on a number of lags [4]. It is closely connected to Box-Pierce test.

The null hypothesis of the Ljung-Box Test,  $H_0$ , is that the data are independently distributed (i.e. the correlations in the population from which the sample is taken are 0, so that any observed correlations in the data result from randomness of the sampling process). In other words, the data is white noise. The alternative hypothesis,  $H_a$ , is that the data are not independently distributed; they exhibit serial correlation [4].

The Ljung-Box test is commonly used in ARIMA modeling. It was applied to the residuals of our fitted ARIMA model to test if residuals from the model have no autocorrelation. From below Ljung-Box test results table we can see that for first 40 lags, p-values are greater than 0.05, indicating residuals are randomly distributed and there is no autocorrelation existed in the residuals. Assuming lags exceeding 40 lags exhibited the same pattern, it is safe to say that the model provided an adequate fit to the data.

	<b>lb_stat</b>	<b>lb_pvalue</b>		<b>lb_stat</b>	<b>lb_pvalue</b>
1	0.14528	0.703088	21	12.927604	0.911139
2	0.154784	0.925527	22	12.941089	0.934791
3	0.21824	0.974592	23	13.017169	0.951617
4	0.252353	0.992679	24	14.585595	0.932301
5	0.253394	0.998429	25	14.680218	0.948541
6	0.321333	0.999387	26	14.970633	0.957869
7	0.33805	0.99985	27	15.272168	0.965439
8	0.650867	0.999639	28	15.935448	0.966766
9	0.670169	0.999894	29	16.690533	0.966772

10	0.67341	0.999973	30	16.764278	0.975295
11	0.696224	0.999992	31	17.752599	0.972579
12	0.699729	0.999998	32	17.833207	0.97954
13	0.726276	0.999999	33	17.870176	0.985186
14	4.991091	0.985936	34	18.007931	0.988851
15	5.549907	0.986358	35	18.058528	0.992028
16	6.282142	0.984751	36	18.201858	0.99407
17	9.793694	0.912038	37	19.528586	0.991877
18	10.453063	0.91611	38	19.561854	0.994236
19	10.884056	0.927677	39	19.585032	0.995973
20	12.859491	0.883331	40	20.09682	0.996361

Table 4 Ljung-Box Test

### 3.7 Model Fitting for MSFT, GOOG and AMZN

Table 5 below shows ARIMA model fitted for MSFT, GOOG and AMZN with their respective MAPE and RMSE scores for comparison with other models. MSFT, GOOG and AMZN stock price forecast plots are included in Appendix A as figure 21-23.

ARIMA	Model	RMSE	MAPE
MSFT	(8, 1, 1)	3.985	10.700%
GOOG	(8, 1, 1)	26.974	9.714%
AMZN	(8, 1, 1)	38.069	8.161%

Table 5 ARIMA model results for MSFT, GOOG and AMZN

## 4. Long Short Term Memory Model

### 4.1 Methodology

LSTM model is a special kind of Recurrent Neural Network (RNN) and was developed to combat the vanishing gradients problem in training traditional RNNs. Unlike standard feedforward neural networks, LSTM has feedback connections which is capable of learning long-term dependencies. LSTM has three gates: the input gate, the forget gate and the output gate. The update gate adds

information to the cell state. The forget gate determines and removes information that is no longer required by the model. The output gate determines the amount of information to output as activations to the next layer. LSTM is introduced by Hochreiter & Schmidhuber (1997). Compared with standard RNN, which has a simple structure with a single tanh layer (figure 8),

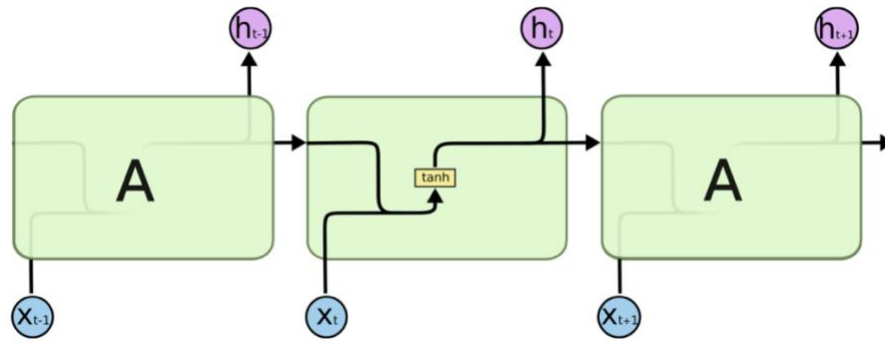


Figure 8 Standard RNN repeating module with single layer [5]

LSTM has four interacting layers in each repeating module in the same chain-like structure (figure 9). In the figure 9, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations [5].

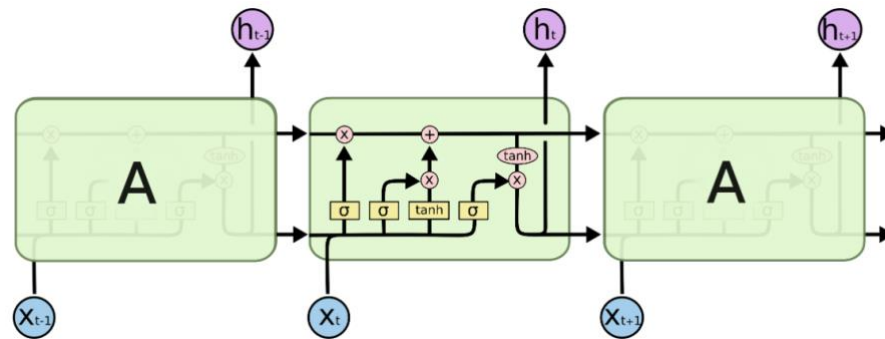


Figure 9 LSTM repeating module with interacting layers [5]

The first step in LSTM is to decide what information would be forgotten from the cell state. This decision is made by a sigmoid layer called “forget gate layer”. It looks at  $h_{t-1}$  and  $x_t$ , and outputs a number between 0 and 1 for each input from cell state  $C_{t-1}$ , where 1 represents “completely keep this” while a 0 represents “completely get rid of this”.  $C_{t-1}$  is then updated into the new cell state  $C_t$ . This process is represented by below formulae [5]:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The old state is then multiplied by  $f_t$ , forgetting the things we decided to forget earlier, then we add new candidate values scales by how much we decided to update each state value,  $i_t \cdot \tilde{C}_t$ .

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad [5]$$

Finally, we run a sigmoid layer which decides what parts of the cell state we are going to output. Then we put the cell state through tanh, which outputs value between -1 and 1, and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t) \quad [5]$$

In our adjusted close price forecast model, we use two layers of LSTM modules, and a dropout layer in-between to avoid over-fitting as shown in figure 10.

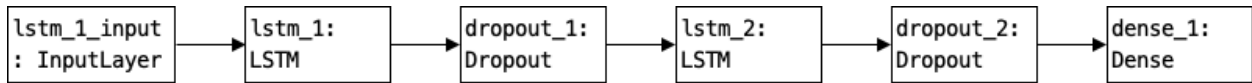


Figure 10 LSTM model flowchart

## 4.2 Model Fitting of VTI stock price

Keras API from TensorFlow package of Python is used to build LSTM model for VTI adjusted close price data, VTI price data was split into 80% training data, 20% cross-validation data and 20% test data. Training dataset was scaled so that each data point has mean zero and standard deviation one. In the scaled training set, we started building LSTM model with initial setting to use past 9 days as features for prediction of today's value. Given that there are 501 observations in the training dataset, the scaled training input and outcome variables has length of 492 and timespan of 9. Cross-validation and test dataset each has length of 166. Train dataset was scaled to have mean zero and standard deviation of one. For each prediction on the development and test dataset, previous N day values, used for features and output, were scaled to have mean 0 and 1. The prediction results were reverted back to original scale by multiplying the previous N day standard deviation and adding back the previous N days mean. RMSE and MAPE scores were calculated between prediction results and the actual stock price at day N.

Initial settings for other LSTM parameters are as follows: dropout probability of 1; LSTM\_Units of 50, where LSTM\_units is the number of hidden units pertinent to the length of state vector, which is also called latent dimension [6]; Optimizer of 'Adam', which is referring to Adam algorithm, a stochastic descent method that is based on adaptive estimation of first-order and second-order moments [7]; epochs of 1 and batch size of 1. After running LSTM model with initial setting on development dataset, we obtained RMSE of 1.985 and MAPE of 1.149% for VTI adjusted close price prediction.

## 4.3 Parameter Tuning

### 1. Tuning N (number of previous values to use as features):

N in the range of 3 to 60 days were fed into LSTM model to find the optimal N which gives lowest RMSE and MAPE on the development dataset. As we can see from figure 11 below, blue line is RMSE score and red line is MAPE (%), optimal N=3 is obtained with lowest RMSE of 1.635 and lowest MAPE of 0.915%. (Table 4.1 of RMSE and MAPE results with different N is listed in Appendix A.)

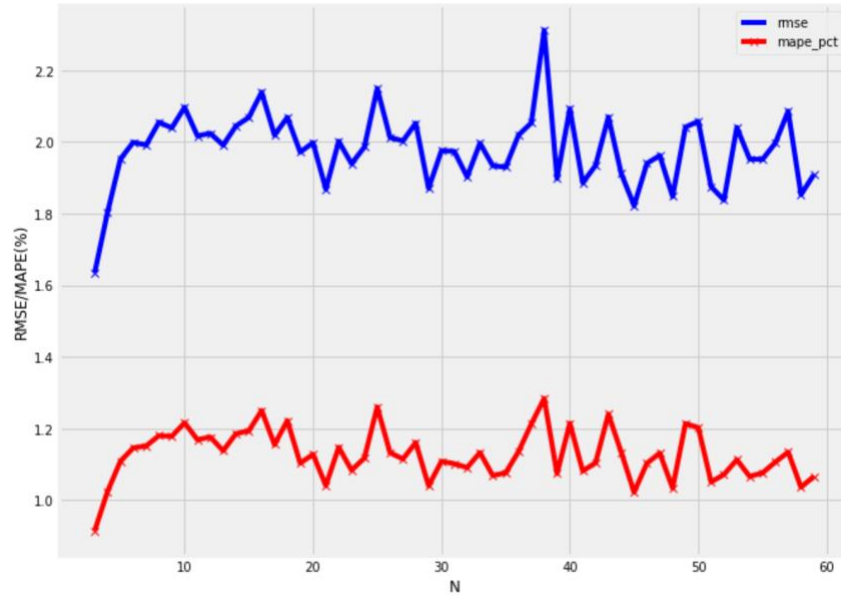


Figure 11 RMSE/MAPE(%) with N

## 2. Tuning epochs and batch\_size

Epochs can be selected from [1, 10, 20, 30, 40, 50] and optimal value for batch\_size can be selected from [8, 16, 32, 64, 128]. We ran LSTM model on development dataset to come up with the optimal parameter setting for epochs and batch\_size based on RMSE and MAPE (%). As can be observed from figure 12 below, batch\_size 8 (the blue line) with epochs of 50 produced the lowest RMSE of 1.247 and lowest MAPE (%) of 0.665%. (Table 4.2 of RMSE and MAPE results with different epochs and batch\_size combinations are shown in Appendix A.)



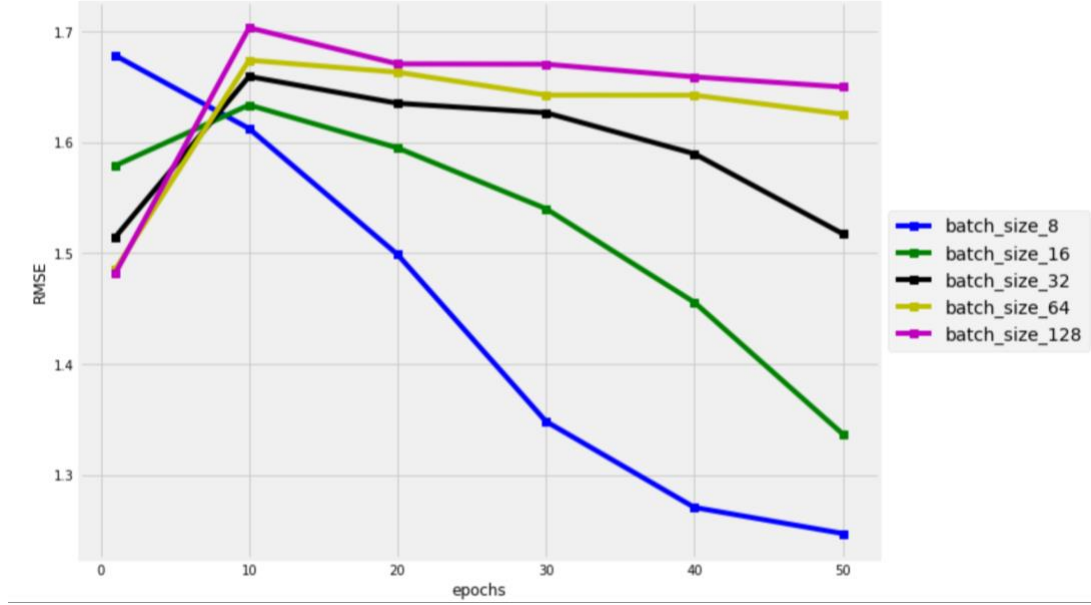


Figure 12 Tuning epochs and batch\_size

### 1. Tuning LSTM units and dropout probability

Optimal LSTM units and dropout probability were selected from [10, 50, 64, 128] and [0.5, 0.6, 0.7, 0.8, 0.9, 1] respectively on the development dataset. As can be observed from figure 13 below, dropout\_prob of 1 (the blue line) with LSTM\_units of 128 produced the lowest RMSE of 1.232 and lowest MAPE (%) of 0.644%. RMSE's decreasing rate slowed down from lstm\_units of 80 with dropout\_prob of 1. (Table 4.3 of RMSE and MAPE results with different LSTM units and dropout probability combinations is shown in Appendix A.)

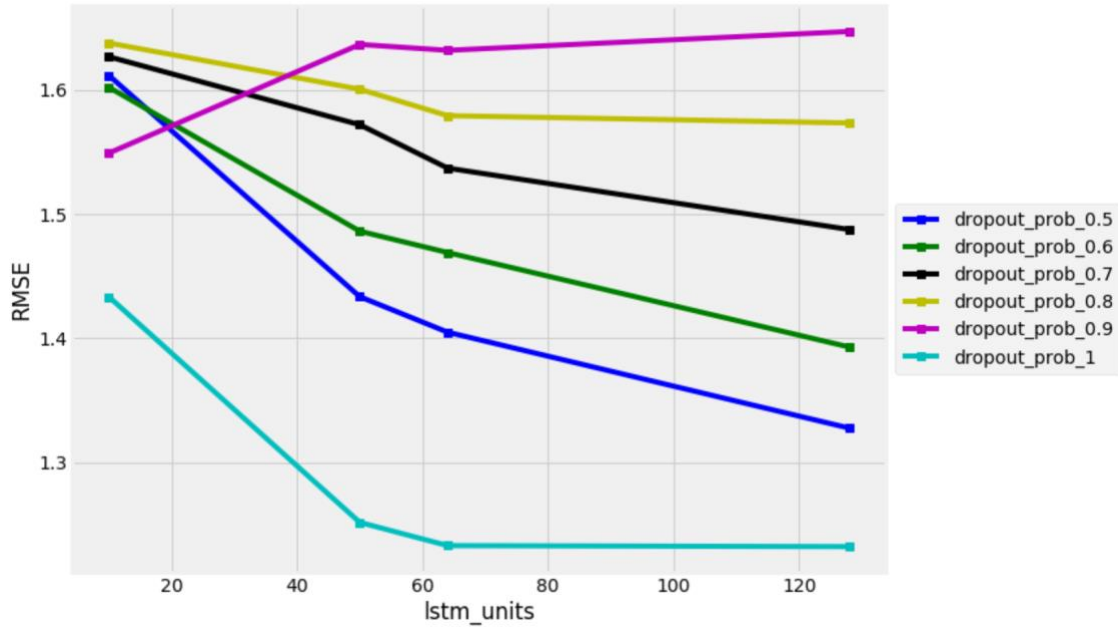


Figure 13 Tuning LSTM units and dropout probability

## 2. Tuning optimizer

Optimal optimizer was selected from ['adam', 'sgd', 'rmsprop', 'adagrad', 'adadelta', 'adamax', 'nadam'] on the development dataset. As can be observed from table 6 below, adamax produced the lowest RMSE of 1.233 and MAPE (%) of 0.651%.

optimizer	rmse	mape_pct
adam	1.235473	0.648086
sgd	1.567141	0.875689
rmsprop	1.233444	0.644605
adagrad	1.27349	0.68625
adadelta	1.240973	0.656432
adamax	1.233297	0.650866
nadam	1.237552	0.644601

Table 6 Tuning optimizer

#### 4.4 Final Model and Prediction results

After tuning, below optimal parameters (table 7) were selected to replace original parameter setting on development dataset. LSTM model was then applied to test dataset. RMSE on test dataset is 3.227 while MAPE on test dataset is 1.336%. As can be observed from figure 14, LSTM was able to predict VTI price more accurately when it is less volatile. There is almost no visual lag before February 2020, but we can see a slight lag in predictions on the plot after February 2020.

param	original	after_tuning
N	9	3
lstm_units	50	128
dropout_prob	1	1
optimizer	adam	adamax
epochs	1	50
batch_size	1	8
rmse	1.98851	1.23946
mape_pct	1.14488	0.6445

Table 7 LSTM parameters after tuning

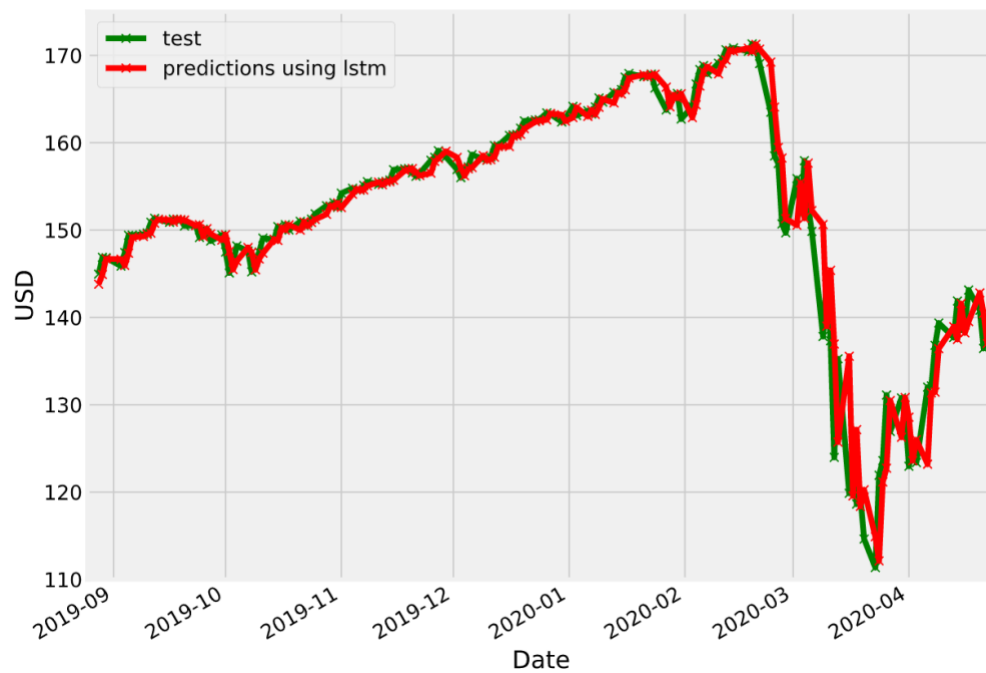


Figure 14 VTI stock price prediction on LSTM test dataset

## 4.5 Model fitting for MSFT, AMZN and GOOG

Table 8 below shows LSTM model fitted for MSFT, GOOG and AMZN with their respective MAPE and RMSE scores. MSFT, GOOG and AMZN stock price forecast price under LSTM are included in Appendix A as figure 24-26.

<b>LSTM</b>	<b>RMSE</b>	<b>MAPE</b>
MSFT	4.278	1.699%
GOOG	29.263	1.525%
AMZN	43.585	1.509%

Table 8 LSTM model results for MSFT, GOOG and AMZN

## 5. Extreme Gradient Boosting Model

### 5.1 Methodology

XGBoost stands for “Extreme Gradient Boosting”, where the term “Gradient Boosting” originates from the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman [8]. Gradient boosting is a process to convert weak learners to strong learners, in an iterative fashion. According to Friedman’s gradient boosting algorithm [8], on each iteration, the gradient descent is first computed in order to fit a new base learner function. Once the best gradient descent step direction and size are found, the function estimation is updated. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models [11]. In pseudocode, the generic gradient boosting method is [8][12]:

Input: training set  $\{(x_i, y_i)\}_{i=1}^2$  a differentiable loss function  $L(y, F(x))$ , number of iterations  $M$ .

Algorithm:

1. Initialize model with a constant value:  $F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$ .

2. For  $m=1$  to  $M$ :

1. Compute so-called pseudo-residuals: for  $i = 1, \dots, n$ ,

$$r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right], \text{ where } F(x) = F_{m-1}(x)$$

2. Fit a base learner (or weak learner, e.g. tree)  $g_m(x)$  to pseudo-residuals, i.e. train it using training set  $\{(x_i, r_{im})\}_{i=1}^n$ .

3. Compute multiplier  $\gamma_m$  by solving the following one-dimensional optimization problem:

$$\gamma_m = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma g_m(x_i))$$

4. Update model:  $F_m(x) = F_{m-1}(x) + \gamma_m g_m(x)$

3. Output  $F_M(x)$ .

XGBoost is an implementation of gradient boosted decision trees algorithm designed for computational speed and model performance, developed by Tianqi Chen [9]. It supports three main gradient boosting methods: gradient boosting machine (GBM) including the learning rate; Stochastic gradient boosting which is the boosting with sub sampling at the row, column and column per split levels; and regularized gradient boosting with L1 or L2 regularization [11]. Different with from GBM which divides the optimization problem into two parts by first determining the direction of the step and then optimizing the step length, XGBoost tries to determine the step directly by solving below equation for each  $x$  in the dataset [13]

$$\frac{\partial L(y, f^{m-1}(x) + f_m(x))}{\partial f_m(x)} = 0 \quad [13]$$

By taking second-order Taylor expansion of the loss function and the current estimate  $f^{m-1}(x)$ , we get  $L(y, f^{m-1}(x) + f_m(x)) \approx L(y, f^{m-1}(x)) + g_m(x)f_m(x) + \frac{1}{2}h_m(x)f_m(x)^2$ , where  $g_m(x)$  is the gradient and  $h_m(x)$  is the Hessian at the current estimate:  $h_m(x) = \frac{\partial^2 L(Y, f(x))}{\partial f(x)^2}$  where  $f(x) = f^{m-1}(x)$ . Then, loss function can be rewritten as

$$L(f_m) \approx \sum_{i=1}^n [g_m(x_i)f_m(x_i) + \frac{1}{2}h_m(x_i)f_m(x_i)^2] + \text{const.}$$

$$\propto \sum_{j=0}^{T_m} \sum_{i \in R_{jm}} [g_m(x_i)w_{jm} + \frac{1}{2}h_m(x_i)w_{jm}^2] \quad [13]$$

Letting  $G_{jm}$  represents the sum of gradient in region  $j$  and  $H_{jm}$  equals to the sum of hessian in region  $j$ , the equation can be rewritten as  $L(f_m) \propto \sum_{j=1}^{T_m} [G_{jm}w_{jm} + \frac{1}{2}H_{jm}w_{jm}^2]$  [13]. With the fixed learned structure, for each region, the optimal weight is  $w_{jm} = -\frac{G_{jm}}{H_{jm}}, j = 1, \dots, T_m$ . Therefore, the loss function is  $L(f_m) \propto -\frac{1}{2} \sum_{j=1}^{T_m} \frac{G_{jm}^2}{H_{jm}}$  [13]. According to Chen [9], this is structure score for a tree.

The smaller the score is, the better the structure is. Thus, for each split made, the proxy gain is defined as  $\text{Gain} = \frac{1}{2} [\frac{G_{jmL}^2}{H_{jmL}} + \frac{G_{jmR}^2}{H_{jmR}} + \frac{G_{jm}^2}{H_{jm}}] = \frac{1}{2} [\frac{G_{jmL}^2}{H_{jmL}} + \frac{G_{jmR}^2}{H_{jmR}} - \frac{(G_{jmL} + G_{jmR})^2}{H_{jmL} + H_{jmR}}]$  [8][13].

Next step is to take regularization into consideration, the loss function becomes below:

$$L(f_m) \propto \sum_{j=0}^{T_m} [G_{jm}w_{jm} + \frac{1}{2}H_{jm}w_{jm}^2] + \gamma T_m + \frac{1}{2}\gamma \sum_{j=1}^{T_m} w_{jm}^2 + \alpha \sum_{j=1}^{T_m} |w_{jm}|$$

$$= \sum_{j=0}^{T_m} [G_{jm}w_{jm} + \frac{1}{2}(H_{jm} + \lambda)w_{jm}^2 + \alpha|w_{jm}|] + \gamma T_m \quad [13]$$

Where  $\gamma$  is the penalization term on the number of terminal nodes,  $\alpha$  and  $\lambda$  are for L1 and L2 regularization respectively. The optimal weight for each region  $j$  is defined as:

$$w_{jm} = \begin{cases} -\frac{G_{jm} + \alpha}{H_{jm} + \lambda}, & G_{jm} < -\alpha \\ -\frac{G_{jm} - \alpha}{H_{jm} + \lambda}, & G_{jm} > \alpha \\ 0, & \text{else.} \end{cases} \quad [13]$$

The gain of each split is defined as [8][13]

$$\text{Gain} = \frac{1}{2} \left[ \frac{T_\alpha(G_{jmL})^2}{H_{jmL} + \lambda} + \frac{T_\alpha(G_{jmR})^2}{H_{jmR} + \lambda} - \frac{T_\alpha(G_{jm})^2}{H_{jm} + \lambda} \right] - \lambda, T_\alpha(G) = \begin{cases} G + \alpha, & G < -\alpha \\ G - \alpha, & G > \alpha \\ 0, & \text{else} \end{cases}$$

## 5.2 Model Development of VTI stock price

The XGBoost model was trained on the training dataset, its parameters were tuned in development dataset and we obtained prediction results on the test dataset. The same as previous LSTM model, VTI adjusted closing price of previous  $N$  days were used as features in the model to predict price on  $N+1$  day. However, in this model, feature engineering has been performed and below 2 additional features were created:

1. range\_hl: the range between high price and low price of the last  $N$  days.
2. range\_oc: the range between open and close price of the last  $N$  days.

Training dataset has been scaled to have mean 0 and standard deviation 1 too. In addition, for development and test dataset, previous  $N$  days' features, such as adjusted closing price, volume, range\_hl etc., were all scaled to have mean 0 and standard deviation 1. It is worth noting here that instead of using the same scaling across training, development and test dataset, we used mean and variance of previous  $N$  days to scale features and predicted values in all datasets. Predictions were later reverted back to their original scale for output and calculation of RMSE and MAPE.

Initial model parameter setting is shown below, where  $n\_estimators = 100$  is number of boosted trees to fit;  $max\_depth = 3$  is maximum tree depth for base learners;  $learning\_rate = 0.1$  is boosting learning rate and  $min\_child\_weight = 1$  is minimum sum of instance weight (hessian) needed in a child;  $gamma = 0$  is minimum loss reduction required to make a further partition on a leaf node of a tree.

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1, colsample_bynode=1,
colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain', interaction_constraints=None,
learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=nan,
monotone_constraints=None, n_estimators=100, n_jobs=0, num_parallel_tree=1,
objective='reg:squarederror', random_state=100, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
seed=100, subsample=1, tree_method=None, validate_parameters=False, verbosity=None)
```

$N = 2$  is used as model input, which means that for features at day  $t$ , we use data from  $t-1$  and  $t-2$  as additional features to be added to the data frame. Below table 9 shows RMSE and MAPE (%) value of different  $N$  on the development dataset.  $N=2$  has the lowest RMSE and MAPE score.

<b>N</b>	<b>RMSE</b>	<b>MAPE</b>
2	1.7523	0.743%
3	1.9542	0.818%
4	2.1785	0.915%
5	2.3740	0.994%
6	2.5735	1.063%
7	2.7357	1.118%
14	3.8686	1.603%

Table 9 RMSE and MAPE (%) for  $N$

Training, development and test dataset were divided into 80% (length 501), 20% (length 166) and 20% (length 166). After feature engineering, additional column of `range_hl` and `range_oc` were added. In the next step, features from previous 2 days were used as features of day 3 as can be seen



from table 10, where lag\_1 used previous day data and lag\_2 used the day before previous day features. Therefore, we have 499 data points in training dataset.

date	adj_close	volume	month	range_hl	range_oc	order_ day	adj_close_ lag_1
1/5/17	109.91947	2604000	1	0.689994	0.089996	2	110.135796
1/6/17	110.26749	2317600	1	0.879998	-0.210006	3	109.919472
1/9/17	109.84421	2461400	1	0.400001	0.319999	4	110.267487
1/10/17	109.92886	2055300	1	0.739998	-0.110001	5	109.844208
1/11/17	110.2769	2751800	1	0.75	-0.329994	6	109.928856
range_hl lag_1	range_oc lag_1	volume_l ag_1	adj_close lag_2	range_hl lag_2	range_oc lag_2	volume lag_2	
0.770004	-0.619995	3228700	109.29867	1.060005	-0.049995	2731600	
0.689994	0.089996	2604000	110.1358	0.770004	-0.619995	3228700	
0.879998	-0.210006	2317600	109.91947	0.689994	0.089996	2604000	
0.400001	0.319999	2461400	110.26749	0.879998	-0.210006	2317600	
0.739998	-0.110001	2055300	109.84421	0.400001	0.319999	2461400	

Table 10 VTI training data

Scaling was performed on training, development and test dataset. Plot 15 below shows how the time series of VTI adjusted closing price compared with training series after scaling.

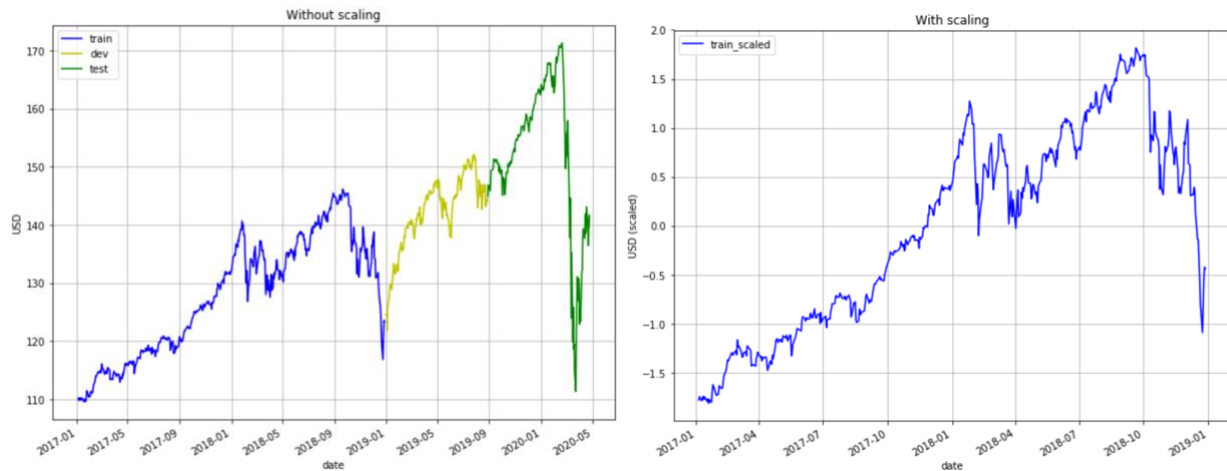


Figure 15 VTI price without scaling and training series with scaling

As can be seen from output below, important features are dominated by adjusted\_close price for the previous N days.

[('volume\_lag\_1', 0.0015100354), ('range\_hl\_lag\_2', 0.0016028018), ('volume\_lag\_2', 0.0018247962), ('range\_oc\_lag\_1', 0.0019524604), ('range\_oc\_lag\_2', 0.0024724023), ('range\_hl\_lag\_1', 0.0032888774), ('adj\_close\_lag\_2', 0.40000784), ('adj\_close\_lag\_1', 0.5873408)]

## 5.3 Parameter Tuning

### 1. Tuning n\_estimators and max\_depth

Candidates for number of estimators (fitted boosted trees) ranged from 10 to 250 with gap of 10. Maximum tree depth were selected from 2 to 9. As can be from seen from figure 16 below, max\_depth of 2 with n\_estimators of 20 achieved the lowest RMSE score. (Table 5.3 of detailed RMSE and MAPE(%) data for each combination of n\_estimators and max\_depth is provided in Appendix A.)

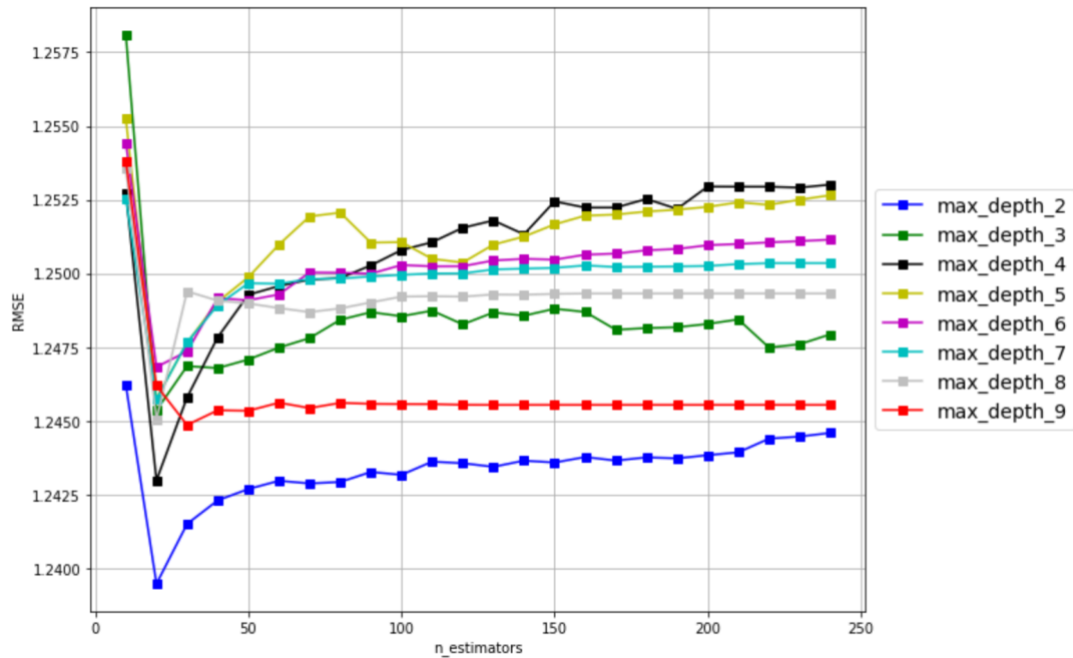


Figure 16 Tuning n\_estimators and max\_depth

### 2. Tuning learning\_rate and min\_child\_weight

Candidates for optimal learning rate were 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3. Minimum child weight can be selected from 5 to 20. Learning\_rate of 0.1 and min\_child\_weight of 5 achieved the lowest RMSE score of 1.239. Table 5.4 of detailed RMSE and MAPE(%) data for each combination of learning\_rate and min\_child\_weight is provided in Appendix A. As can be seen from table 5.4, learning rate is the main reason cause RMSE and MAPE (%) score to change. Various level of child weight does not change forecast accuracy notably for the same learning rate.

### 3. Tuning subsample and gamma parameters

Subsample ratio of the training instance can be selected from 0.1 to 1 and gamma can be selected from 0 to 0.8. Parameter setting of subsample = 0.1 and gamma = 0.3 (yellow line) yielded the lowest RMSE of 1.237, which can be seen from plot 18 below and table 5.5 in Appendix A.

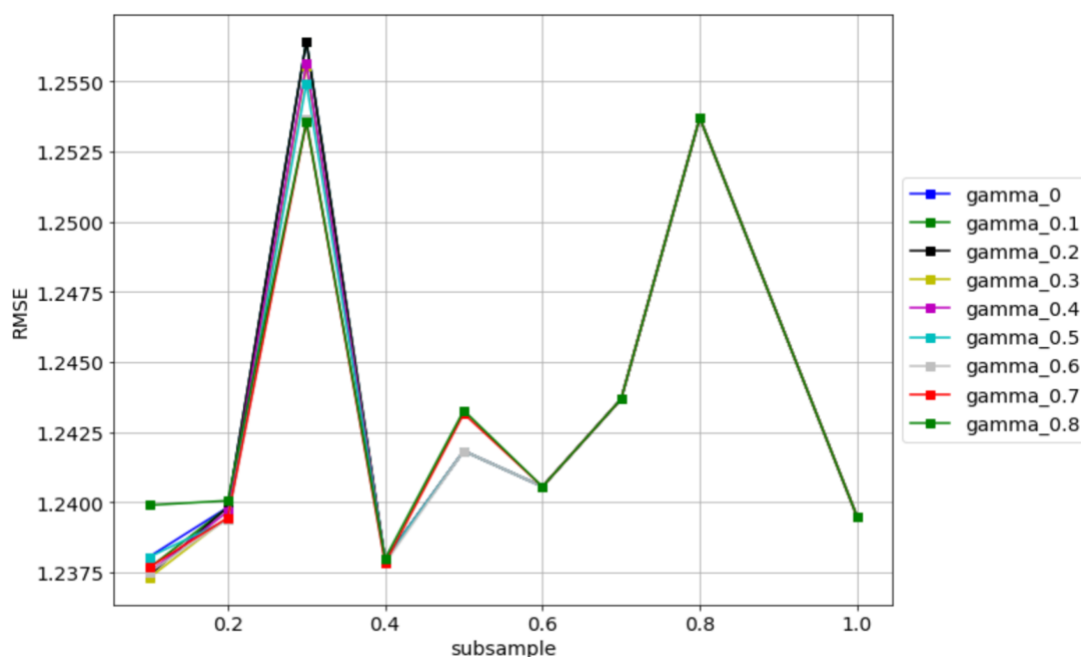


Figure 17 Tuning subsample and gamma parameters

### 4. Tuning colsample\_bytree and colsample\_bylevel

Colsample\_bytree is the subsample ratio of columns when constructing each tree. Colsample\_bylevel is the subsample ratio of columns for each split in each level. For both parameters, candidate values ranged from 0.5 to 1. As can be observed from plot 18, lowest RMSE of 1.237 was achieved with initial setting of colsample\_bytree = 1 and colsample\_bylevel = 1. (Table 5.6 with detailed information of RMSE and MAPE (%) scores for different combinations of colsample\_bytree and colsample\_bylevel is provided in Appendix A.)

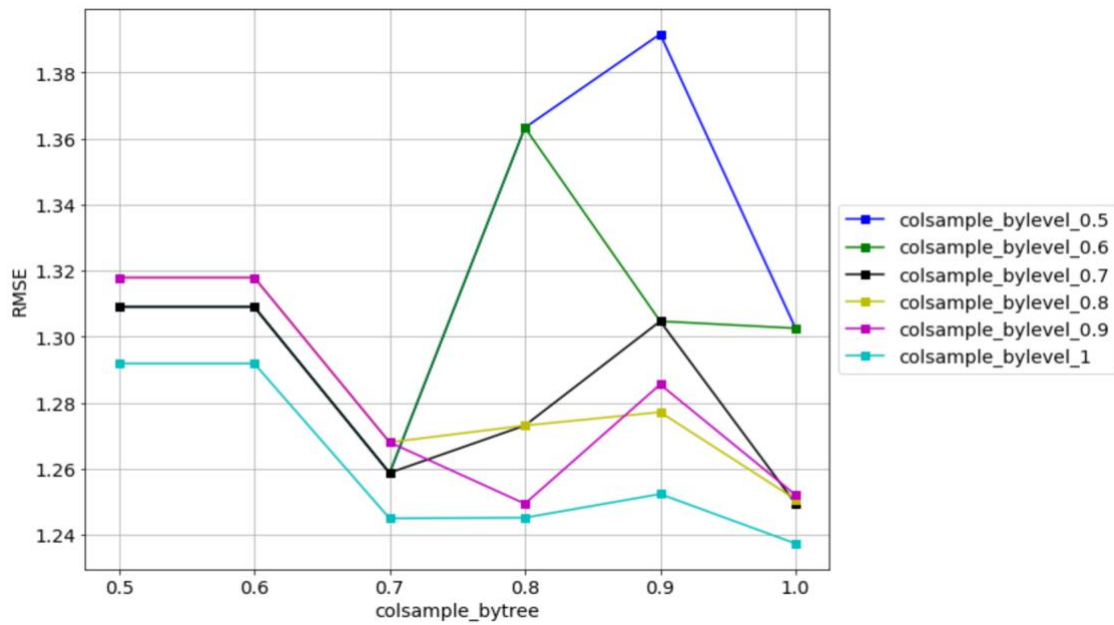


Figure 18 Tuning colsample\_bytree and colsample\_bylevel

## 5.4 Final Model and Prediction results

After tuning, below optimal parameters (table 11) were selected on development dataset to replace original parameter setting. XGBoost model was ran on test dataset. RMSE on test dataset is 3.080 while MAPE on test dataset is 1.274%. This result is better than RMSE and MAPE (%) scores obtained in LSTM. As can be observed from prediction plot figure 19, XGBoost's prediction of VTI price is quite accurate. Prediction produced by XGBoost is more accurately when it is less

volatile. There is almost no visual lag before February 2020, but we can see a slight lag in predictions on the plot after February 2020.

param	original	after_tuning
n_estimators	100	20
max_depth	3	2
learning_rate	0.1	0.1
min_child_weight	1	5
subsample	1	0.1
colsample_bytree	1	1
colsample_bylevel	1	1
gamma	0	0.3
rmse	1.249	1.237
mape_pct	0.665	0.650

Table 11 XGBoost parameters after tuning

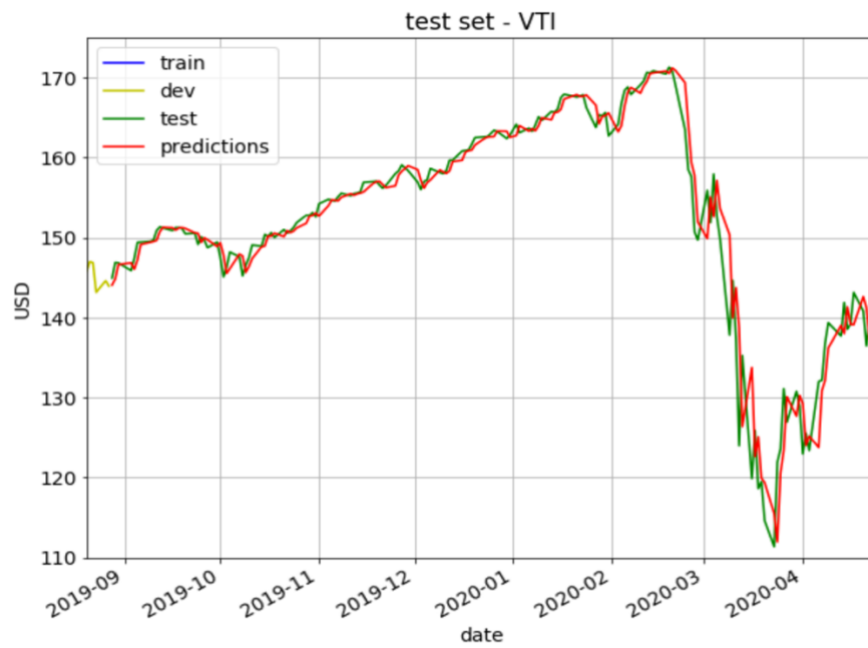


Figure 19 VTI stock price prediction on XGBoost test dataset

## 5.5 Model Fitting for MSFT, AMZN and GOOG

Table 12 below shows XGBoost model fitted for MSFT, GOOG and AMZN with their respective MAPE and RMSE scores. MSFT, GOOG and AMZN stock price forecast price under XGBoost are included in Appendix A as figure 27-29.

<b>XGBoost</b>	<b>RMSE</b>	<b>MAPE</b>
MSFT	4.145	1.624%
GOOG	27.910	1.449%
AMZN	39.427	1.380%

Table 12 XGBoost model results for MSFT, GOOG and AMZN

## 6. Conclusion and Recommendation

### 6.1 Conclusion

<b>VTI</b>	<b>RMSE</b>	<b>MAPE</b>
ARIMA	2.985	9.675%
LSTM	3.277	1.336%
XGBoost	3.080	1.274%
<b>MSFT</b>	<b>RMSE</b>	<b>MAPE</b>
ARIMA	3.985	10.700%
LSTM	4.278	1.699%
XGBoost	4.145	1.624%
<b>GOOG</b>	<b>RMSE</b>	<b>MAPE</b>
ARIMA	26.974	9.714%
LSTM	29.263	1.525%
XGBoost	27.910	1.449%
<b>AMZN</b>	<b>RMSE</b>	<b>MAPE</b>
ARIMA	38.069	8.161%
LSTM	43.585	1.509%
XGBoost	39.427	1.380%

Table 13 Model Comparison

In above RMSE and MAPE (%) comparison table for all models, we can see that although ARIMA model outperformed other machine learning models in RMSE, its MAPE (%) are much bigger in comparison with other models. Based on RMSE values, it is sufficient to conclude that all three models can provide good predictions of future stock price. Among them, ARIMA outperforms other models in the respect of achieving lowest RMSE with the trade-off of computing time.

In terms of computing time spent on parameter searching and prediction calculation, XGBoost was the fastest one while ARIMA model was the slowest one as for each value, ARIMA model was fitted again. As for LSTM model, take tuning parameter process as an example, LSTM took around 3-4 minutes to perform calculation while time taken to tune hyperparameter under XGBoost was usually around a fraction of a minute.

In conclusion, we can say that using historical  $N$  days stock price on its own can provide a relatively accurate prediction on  $N+1$  day's stock price. In XGBoost model particularly, we found out that using  $N=2$  provides better RMSE and MAPE(%) results than other larger values of  $N$  (previous  $N$  days). As  $N$  gets larger, prediction accuracy gets lower in XGBoost. In XGBoost feature importance analysis, we found out that the most important factor to today's stock price is its price yesterday.

## 6.2 Recommendation

Grid search on  $(p, d, q)$  of ARIMA model was only limited to a few candidates in chapter 3, however, it takes extensive computing power and longest time to come up with results. To improve the computing efficiency, better grid search or random search for parameter settings of rolling forecast ARIMA model can be conducted.

In addition, other data source can be used to improve the model forecasting accuracy too. 2013 Nobel Prize in Economics winner, Robert Shiller [15]'s research supported the belief that the financial markets are frequently irrational, which in turn gave a boost to the behavioral finance wing of the finance profession. Volatility in the stock market is greatly affected by market sentiment and publicly available information. To look further into the stock price forecast, we can also include behavioral analysis, such as text analysis of news into further research to assess market sentiment and predict market trend.



## Appendix A Figures and Tables

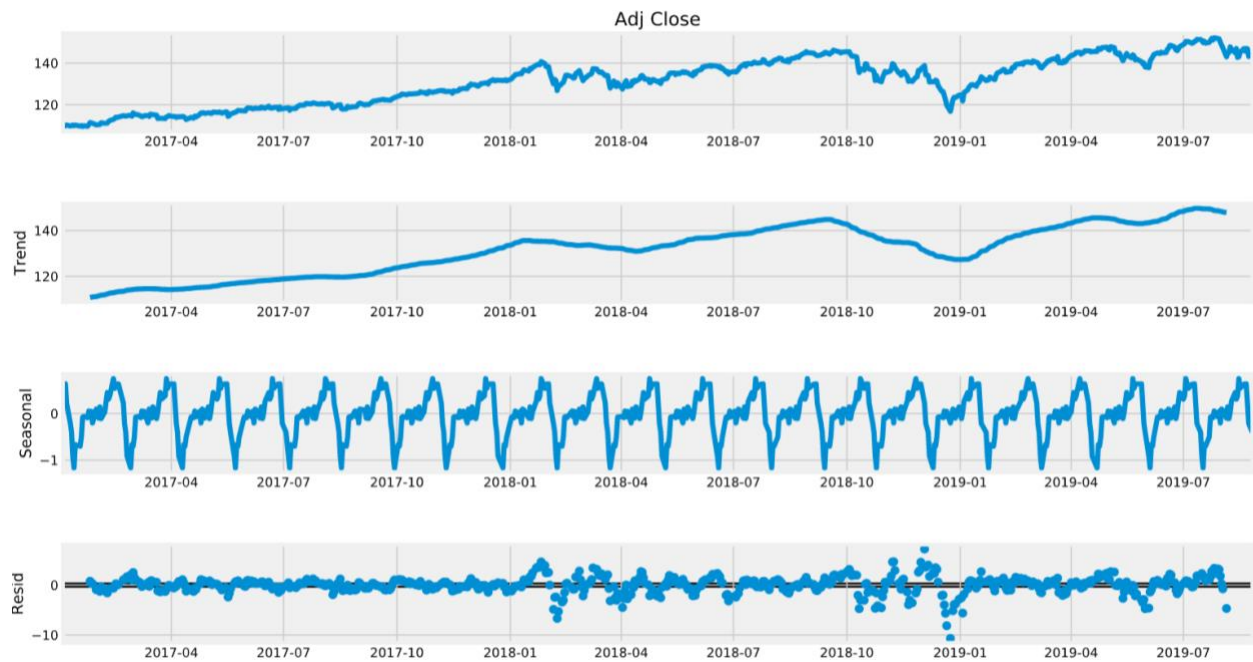


Figure 20 Time Series Decomposition of VTI train data

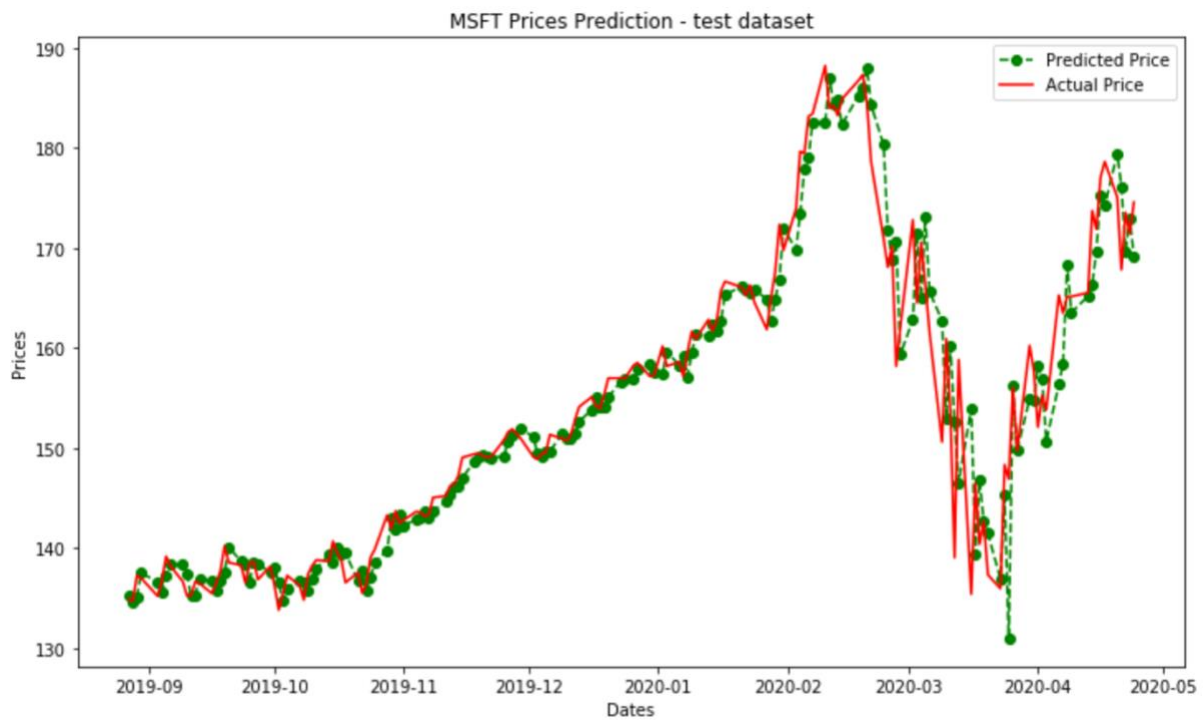


Figure 21 MSFT stock price forecast using ARIMA

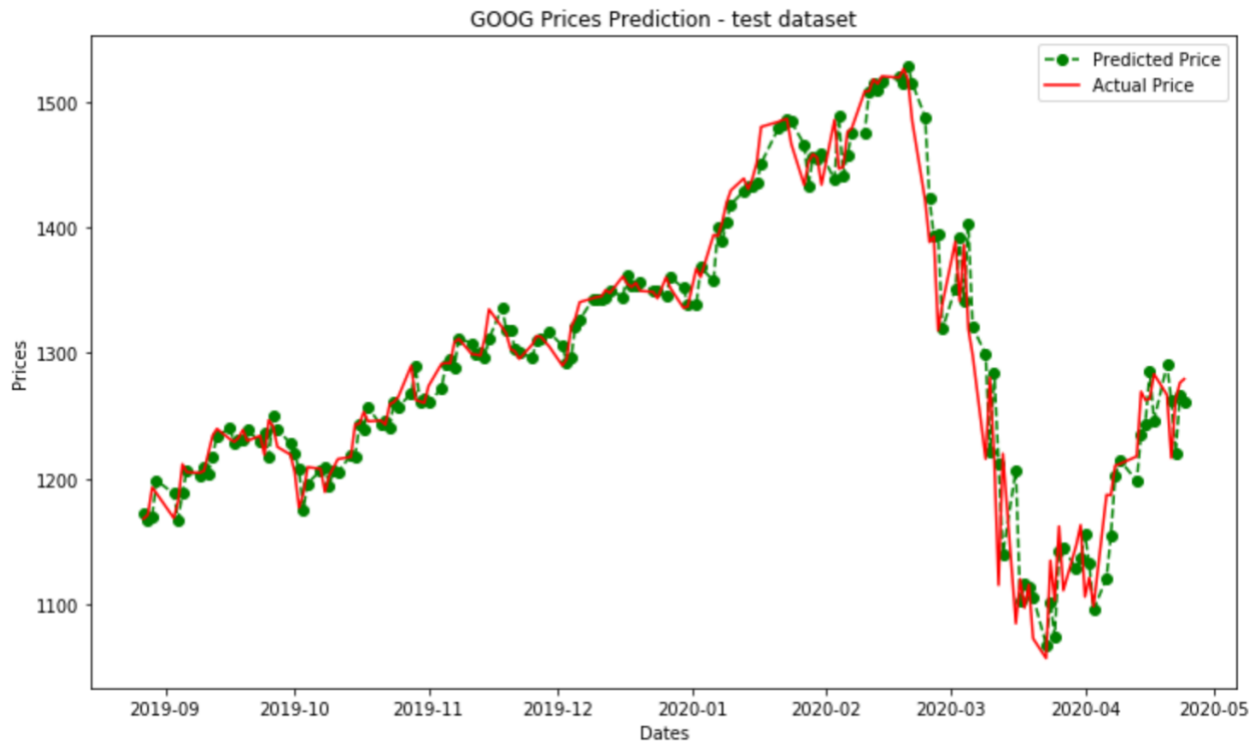


Figure 22 GOOG stock price forecast using ARIMA

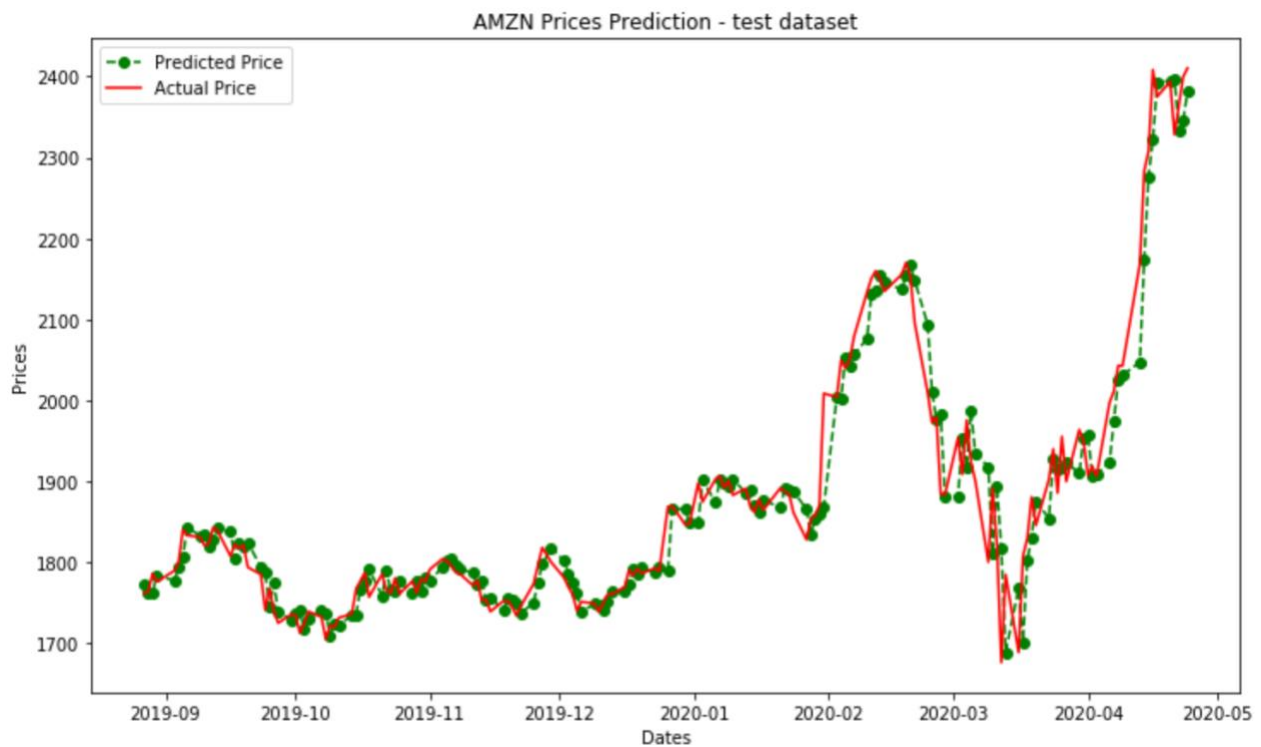


Figure 23 AMZN stock price forecast using ARIMA

ARIMA(0, 0, 0)	RMSE=	22.457	MAPE=	12.947
ARIMA(0, 0, 1)	RMSE=	11.978	MAPE=	9.513
ARIMA(0, 1, 0)	RMSE=	3.199	MAPE=	9.665
ARIMA(0, 1, 1)	RMSE=	3.136	MAPE=	9.667
ARIMA(0, 1, 2)	RMSE=	3.093	MAPE=	9.663
ARIMA(0, 2, 0)	RMSE=	5.232	MAPE=	9.841
ARIMA(0, 2, 1)	RMSE=	3.276	MAPE=	9.709
ARIMA(0, 2, 2)	RMSE=	3.217	MAPE=	9.772
ARIMA(1, 0, 0)	RMSE=	3.192	MAPE=	9.653
ARIMA(1, 0, 1)	RMSE=	3.132	MAPE=	9.656
ARIMA(1, 0, 2)	RMSE=	3.085	MAPE=	9.651
ARIMA(1, 1, 0)	RMSE=	3.114	MAPE=	9.667
ARIMA(1, 1, 1)	RMSE=	3.128	MAPE=	9.645
ARIMA(1, 1, 2)	RMSE=	3.117	MAPE=	9.673
ARIMA(1, 2, 0)	RMSE=	3.658	MAPE=	9.774
ARIMA(1, 2, 1)	RMSE=	3.228	MAPE=	9.767
ARIMA(1, 2, 2)	RMSE=	3.263	MAPE=	9.749
ARIMA(2, 0, 0)	RMSE=	3.11	MAPE=	9.656
ARIMA(2, 0, 1)	RMSE=	3.131	MAPE=	9.651
ARIMA(2, 0, 2)	RMSE=	3.138	MAPE=	9.661
ARIMA(2, 1, 0)	RMSE=	3.124	MAPE=	9.66
ARIMA(2, 1, 1)	RMSE=	3.2	MAPE=	9.691
ARIMA(2, 1, 2)	RMSE=	3.261	MAPE=	9.692
ARIMA(2, 2, 0)	RMSE=	3.395	MAPE=	9.784
ARIMA(2, 2, 1)	RMSE=	3.173	MAPE=	9.74
ARIMA(2, 2, 2)	RMSE=	3.225	MAPE=	9.763
ARIMA(4, 0, 0)	RMSE=	3.126	MAPE=	9.662
ARIMA(4, 0, 1)	RMSE=	3.036	MAPE=	9.646
ARIMA(4, 0, 2)	RMSE=	3.113	MAPE=	9.672
ARIMA(4, 1, 0)	RMSE=	3.161	MAPE=	9.676
ARIMA(4, 1, 1)	RMSE=	3.027	MAPE=	9.652
ARIMA(4, 1, 2)	RMSE=	3.122	MAPE=	9.699
ARIMA(4, 2, 0)	RMSE=	3.349	MAPE=	9.788
ARIMA(4, 2, 1)	RMSE=	3.285	MAPE=	9.717
ARIMA(4, 2, 2)	RMSE=	3.074	MAPE=	9.733
ARIMA(6, 0, 0)	RMSE=	3.191	MAPE=	9.659
ARIMA(6, 0, 2)	RMSE=	3.069	MAPE=	9.669
ARIMA(6, 1, 0)	RMSE=	3.176	MAPE=	9.668
ARIMA(6, 1, 1)	RMSE=	3.018	MAPE=	9.664
ARIMA(6, 1, 2)	RMSE=	3.01	MAPE=	9.669
ARIMA(6, 2, 0)	RMSE=	3.123	MAPE=	9.791
ARIMA(6, 2, 1)	RMSE=	3.253	MAPE=	9.719
ARIMA(8, 0, 0)	RMSE=	3.042	MAPE=	9.68
ARIMA(8, 0, 1)	RMSE=	2.985	MAPE=	9.675
ARIMA(8, 0, 2)	RMSE=	3.025	MAPE=	9.677
ARIMA(8, 1, 0)	RMSE=	3.033	MAPE=	9.671
ARIMA(8, 1, 1)	RMSE=	3.004	MAPE=	9.678
ARIMA(8, 2, 0)	RMSE=	3.047	MAPE=	9.79
ARIMA(8, 2, 1)	RMSE=	3.115	MAPE=	9.747

Table 14 ARIMA model Grid search RMSE and MAPE results

ARIMA(0, 0, 0)x(0, 0, 0, 30)30 - AIC:8382.097997996378  
 ARIMA(0, 0, 0)x(0, 0, 1, 30)30 - AIC:7259.843990374895  
 ARIMA(0, 0, 0)x(0, 1, 0, 30)30 - AIC:3967.7552518929624  
 ARIMA(0, 0, 0)x(0, 1, 1, 30)30 - AIC:3796.0168574823565  
 ARIMA(0, 0, 0)x(1, 0, 0, 30)30 - AIC:3918.9819327066425  
 ARIMA(0, 0, 0)x(1, 0, 1, 30)30 - AIC:3897.966040052399  
 ARIMA(0, 0, 0)x(1, 1, 0, 30)30 - AIC:3801.3312634737426  
 ARIMA(0, 0, 0)x(1, 1, 1, 30)30 - AIC:3797.942633046885  
 ARIMA(0, 0, 1)x(0, 0, 0, 30)30 - AIC:7462.298688694405  
 ARIMA(0, 0, 1)x(0, 0, 1, 30)30 - AIC:6382.914039721617  
 ARIMA(0, 0, 1)x(0, 1, 0, 30)30 - AIC:3271.0854229071692  
 ARIMA(0, 0, 1)x(0, 1, 1, 30)30 - AIC:3128.1964490774694  
 ARIMA(0, 0, 1)x(1, 0, 0, 30)30 - AIC:3237.885178377963  
 ARIMA(0, 0, 1)x(1, 0, 1, 30)30 - AIC:3197.4325675079263  
 ARIMA(0, 0, 1)x(1, 1, 0, 30)30 - AIC:3139.070064576642  
 ARIMA(0, 0, 1)x(1, 1, 1, 30)30 - AIC:3130.1964491434856  
 ARIMA(0, 1, 0)x(0, 0, 0, 30)30 - AIC:2020.9634485376653  
 ARIMA(0, 1, 0)x(0, 0, 1, 30)30 - AIC:1956.2985079273822  
 ARIMA(0, 1, 0)x(0, 1, 0, 30)30 - AIC:2360.9998082417537  
 ARIMA(0, 1, 0)x(0, 1, 1, 30)30 - AIC:1931.183327946681  
 ARIMA(0, 1, 0)x(1, 0, 0, 30)30 - AIC:1958.385587362306  
 ARIMA(0, 1, 0)x(1, 0, 1, 30)30 - AIC:1958.298507520596  
 ARIMA(0, 1, 0)x(1, 1, 0, 30)30 - AIC:2117.2854653158815  
 ARIMA(0, 1, 0)x(1, 1, 1, 30)30 - AIC:1949.1878868892563  
 ARIMA(0, 1, 1)x(0, 0, 0, 30)30 - AIC:2020.4901490089765  
 ARIMA(0, 1, 1)x(0, 0, 1, 30)30 - AIC:1955.774609301062  
 ARIMA(0, 1, 1)x(0, 1, 0, 30)30 - AIC:2359.288017266959  
 ARIMA(0, 1, 1)x(0, 1, 1, 30)30 - AIC:1930.4592346412535  
 ARIMA(0, 1, 1)x(1, 0, 0, 30)30 - AIC:1959.9628870614156  
 ARIMA(0, 1, 1)x(1, 0, 1, 30)30 - AIC:1957.7746061721566  
 ARIMA(0, 1, 1)x(1, 1, 0, 30)30 - AIC:2119.1629497693953  
 ARIMA(0, 1, 1)x(1, 1, 1, 30)30 - AIC:1947.408887377065  
 ARIMA(1, 0, 0)x(0, 0, 0, 30)30 - AIC:2024.4010380926918  
 ARIMA(1, 0, 0)x(0, 0, 1, 30)30 - AIC:1960.7308181776634  
 ARIMA(1, 0, 0)x(0, 1, 0, 30)30 - AIC:2353.600418615331  
 ARIMA(1, 0, 0)x(0, 1, 1, 30)30 - AIC:1933.0323362551794  
 ARIMA(1, 0, 0)x(1, 0, 0, 30)30 - AIC:1959.4782700476935  
 ARIMA(1, 0, 0)x(1, 0, 1, 30)30 - AIC:1961.478329009712  
 ARIMA(1, 0, 0)x(1, 1, 0, 30)30 - AIC:2111.462297276691  
 ARIMA(1, 0, 0)x(1, 1, 1, 30)30 - AIC:1951.0942408119433  
 ARIMA(1, 0, 1)x(0, 0, 0, 30)30 - AIC:2023.3771739264416  
 ARIMA(1, 0, 1)x(0, 0, 1, 30)30 - AIC:1961.1753979987861  
 ARIMA(1, 0, 1)x(0, 1, 0, 30)30 - AIC:2350.345987868199  
 ARIMA(1, 0, 1)x(0, 1, 1, 30)30 - AIC:1932.3939519048924  
 ARIMA(1, 0, 1)x(1, 0, 0, 30)30 - AIC:1960.9837754327164  
 ARIMA(1, 0, 1)x(1, 0, 1, 30)30 - AIC:1960.8942293275472  
 ARIMA(1, 0, 1)x(1, 1, 0, 30)30 - AIC:2113.462231598986  
 ARIMA(1, 0, 1)x(1, 1, 1, 30)30 - AIC:1949.5629926541435  
 ARIMA(1, 1, 0)x(0, 0, 0, 30)30 - AIC:2022.5885814606213  
 ARIMA(1, 1, 0)x(0, 0, 1, 30)30 - AIC:1957.9054528642034

ARIMA(1, 1, 0)x(0, 1, 0, 30)30 - AIC:2362.2366869287484  
 ARIMA(1, 1, 0)x(0, 1, 1, 30)30 - AIC:1933.0224370443543  
 ARIMA(1, 1, 0)x(1, 0, 0, 30)30 - AIC:1957.906623920508  
 ARIMA(1, 1, 0)x(1, 0, 1, 30)30 - AIC:1959.9053762215815  
 ARIMA(1, 1, 0)x(1, 1, 0, 30)30 - AIC:2116.5117962417958  
 ARIMA(1, 1, 0)x(1, 1, 1, 30)30 - AIC:1950.3104529744055  
 ARIMA(1, 1, 1)x(0, 0, 0, 30)30 - AIC:2017.5417805713964  
 ARIMA(1, 1, 1)x(0, 0, 1, 30)30 - AIC:1952.1177509933982  
 ARIMA(1, 1, 1)x(0, 1, 0, 30)30 - AIC:2343.507671597427  
 ARIMA(1, 1, 1)x(0, 1, 1, 30)30 - AIC:1925.8997424010618  
 ARIMA(1, 1, 1)x(1, 0, 0, 30)30 - AIC:1954.7109916704085  
 ARIMA(1, 1, 1)x(1, 0, 1, 30)30 - AIC:1954.1176476180492  
 ARIMA(1, 1, 1)x(1, 1, 0, 30)30 - AIC:2112.311316308107  
 ARIMA(1, 1, 1)x(1, 1, 1, 30)30 - AIC:1949.1576401421905

Table 15 SARIMA(p, d, q) x (P, D, Q, S) grid search output

Table 4.1 Tuning N in LSTM

N	rmse	mape_pct
3	1.635137	0.914945
4	1.803699	1.024476
5	1.952342	1.108799
6	1.999262	1.146063
7	1.99164	1.151408
8	2.055203	1.180261
9	2.03939	1.178511
10	2.096836	1.216154
11	2.017165	1.169035
12	2.024144	1.176395
13	1.991401	1.138119
14	2.044767	1.185701
15	2.068667	1.193856
16	2.138989	1.250351
17	2.019324	1.155408
18	2.068833	1.222217
19	1.971816	1.103459
20	1.998074	1.128057
21	1.868563	1.041144
22	2.002149	1.147849
23	1.939465	1.083878
24	1.985929	1.117767
25	2.149663	1.260305
26	2.012484	1.132606
27	2.00327	1.115186
28	2.052448	1.160969
29	1.870812	1.040309
30	1.975864	1.108656
31	1.973921	1.10194

Table 4.2 Tuning epochs and batch size in LSTM

epochs	batch_size	rmse	mape_pct
1	8	1.678135	0.938969
1	16	1.579001	0.881897
1	32	1.514556	0.843834
1	64	1.485473	0.825802
1	128	1.481985	0.824164
10	8	1.612337	0.901145
10	16	1.633598	0.912933
10	32	1.659342	0.927644
10	64	1.674018	0.936551
10	128	1.703296	0.954901
20	8	1.498725	0.836228
20	16	1.594895	0.891542
20	32	1.635131	0.913929
20	64	1.663155	0.929925
20	128	1.670698	0.934552
30	8	1.348068	0.741547
30	16	1.540042	0.859702
30	32	1.626615	0.909416
30	64	1.642572	0.91812
30	128	1.670414	0.934289
40	8	1.270451	0.684208
40	16	1.455515	0.812535
40	32	1.58958	0.888569
40	64	1.642431	0.917835
40	128	1.65896	0.927417
50	8	1.246896	0.665029
50	16	1.336626	0.733488
50	32	1.517376	0.846982
50	64	1.625313	0.908454

32	1.90275	1.091215
33	1.996743	1.133187
34	1.93373	1.068776
35	1.929883	1.076478
36	2.019225	1.135632
37	2.053563	1.213014
38	2.311833	1.283955
39	1.898322	1.074329
40	2.092808	1.214837
41	1.886248	1.08144
42	1.934443	1.104619
43	2.069917	1.240189
44	1.911959	1.134062
45	1.822352	1.022273
46	1.940279	1.103122
47	1.962257	1.131563
48	1.848105	1.033982
49	2.041646	1.213759
50	2.05762	1.202696
51	1.875312	1.050991
52	1.837912	1.071799
53	2.040788	1.112964
54	1.952476	1.066455
55	1.95152	1.076401
56	1.997061	1.10647
57	2.08617	1.134305
58	1.853241	1.036441
59	1.909123	1.06632

Table 16 Tuning N in LSTM; Tuning epochs and batch\_size in LSTM; Tuning LSTM\_units and dropout\_prob

50	128	1.649868	0.921989
----	-----	----------	----------

Table 4.3 Tuning LSTM units and dropout\_prob

lstm_units	dropout_prob	rmse	mape_pct
10	0.5	1.611825	0.899506
10	0.6	1.602218	0.895532
10	0.7	1.626824	0.909255
10	0.8	1.637885	0.916009
10	0.9	1.549243	0.864072
10	1	1.433534	0.797652
50	0.5	1.433589	0.796013
50	0.6	1.486345	0.827256
50	0.7	1.572073	0.878883
50	0.8	1.600658	0.894529
50	0.9	1.636746	0.914418
50	1	1.251832	0.669152
64	0.5	1.404998	0.77916
64	0.6	1.469167	0.817577
64	0.7	1.537158	0.858262
64	0.8	1.579387	0.882026
64	0.9	1.631984	0.911581
64	1	1.233128	0.652146
128	0.5	1.327776	0.721957
128	0.6	1.393073	0.769651
128	0.7	1.487709	0.828222
128	0.8	1.573512	0.877516
128	0.9	1.647166	0.919739
128	1	1.232318	0.643946

Table 5.3 Tuning n\_estimators and max\_depth

n_estimators	max_depth	rmse	mape_pct
10	2	1.246225	0.657427
10	3	1.258097	0.667729
10	4	1.252735	0.662561
10	5	1.255246	0.665594
10	6	1.254419	0.664952
10	7	1.25254	0.662556
10	8	1.253544	0.66318
10	9	1.253802	0.663246
20	2	1.239485	0.653097
20	3	1.245364	0.660903
20	4	1.242989	0.658172
20	5	1.245771	0.661318

Table 5.4 Tuning learning\_rate and min\_child\_weight

learning_rate	min_child_weight	rmse	mape_pct
0.001	5	1.368004	0.732266
0.001	7	1.368004	0.732266
0.001	9	1.368004	0.732266
0.001	11	1.368004	0.732266
0.001	13	1.368004	0.732266
0.001	15	1.368004	0.732266
0.001	17	1.368004	0.732266
0.001	19	1.368004	0.732266
0.005	5	1.356137	0.728705
0.005	7	1.356137	0.728705
0.005	9	1.356137	0.728705
0.005	11	1.356137	0.728705

20	6	1.246832	0.662261
20	7	1.245744	0.660342
20	8	1.245055	0.659495
20	9	1.2462	0.660592
30	2	1.241514	0.652705
30	3	1.246863	0.662874
30	4	1.245804	0.661525
30	5	1.247654	0.663604
30	6	1.24736	0.662934
30	7	1.247671	0.662256
30	8	1.249396	0.664085
30	9	1.244868	0.658698
40	2	1.242312	0.653045
40	3	1.246795	0.66258
40	4	1.247842	0.664058
40	5	1.249044	0.664725
40	6	1.249165	0.664556
40	7	1.248929	0.66329
...	...	...	...
210	4	1.252942	0.66891
210	5	1.252412	0.66768
210	6	1.251002	0.665852
210	7	1.25032	0.665353
210	8	1.249324	0.664059
210	9	1.245549	0.659434
220	2	1.244403	0.655604
220	3	1.247493	0.66401
220	4	1.252942	0.668901
220	5	1.252328	0.667605
220	6	1.251056	0.665894
220	7	1.250351	0.66538
220	8	1.249324	0.664059
220	9	1.245549	0.659434
230	2	1.244474	0.655721
230	3	1.247604	0.664225
230	4	1.252912	0.668857
230	5	1.252502	0.667844
230	6	1.251099	0.665947
230	7	1.250351	0.665381
230	8	1.249324	0.664059
230	9	1.245549	0.659434
240	2	1.244599	0.655668
240	3	1.247939	0.664419
240	4	1.253013	0.668865
240	5	1.252653	0.667909
240	6	1.251149	0.665963
240	7	1.250351	0.665381
240	8	1.249324	0.664059

0.005	13	1.356137	0.728705
0.005	15	1.356137	0.728705
0.005	17	1.356137	0.728705
0.005	19	1.356137	0.728705
0.01	5	1.338376	0.72142
0.01	7	1.338376	0.72142
0.01	9	1.338376	0.72142
0.01	11	1.338376	0.72142
0.01	13	1.338376	0.72142
0.01	15	1.338376	0.72142
0.01	17	1.338376	0.72142
0.01	19	1.338376	0.72142
0.05	5	1.249528	0.65995
0.05	7	1.249528	0.65995
0.05	9	1.249528	0.65995
0.05	11	1.249528	0.65995
0.05	13	1.249528	0.65995
0.05	15	1.249528	0.65995
0.05	17	1.249528	0.65995
0.05	19	1.249528	0.65995
0.1	5	1.239485	0.653097
0.1	7	1.239485	0.653097
0.1	9	1.239485	0.653097
0.1	11	1.239485	0.653097
0.1	13	1.239485	0.653097
0.1	15	1.239485	0.653097
0.1	17	1.239485	0.653097
0.1	19	1.239485	0.653097
0.2	5	1.242186	0.648993
0.2	7	1.242186	0.648993
0.2	9	1.242186	0.648993
0.2	11	1.242186	0.648993
0.2	13	1.242186	0.648993
0.2	15	1.242181	0.648988
0.2	17	1.242203	0.648997
0.2	19	1.24219	0.648988
0.3	5	1.245637	0.648263
0.3	7	1.245839	0.648455
0.3	9	1.245369	0.647755
0.3	11	1.245533	0.647859
0.3	13	1.246013	0.648456
0.3	15	1.246305	0.648464
0.3	17	1.245944	0.648074
0.3	19	1.246517	0.648759

240	9	1.245549	0.659434
-----	---	----------	----------

Table 17 Tuning n\_estimators and max\_depth; Tuning learning\_rate and min\_child\_weight

Table 5.5 Tuning subsample and gamma parameters

subsample	gamma	rmse	mape_pct
0.1	0	1.238055	0.65161
0.1	0.1	1.237653	0.651075
0.1	0.2	1.237387	0.6509
0.1	0.3	1.237304	0.650484
0.1	0.4	1.237501	0.650418
0.1	0.5	1.238046	0.651427
0.1	0.6	1.2375	0.650672
0.1	0.7	1.237678	0.650614
0.1	0.8	1.239896	0.653925
0.2	0	1.23982	0.653798
0.2	0.1	1.23982	0.653798
0.2	0.2	1.23982	0.653798
0.2	0.3	1.23946	0.653353
0.2	0.4	1.239672	0.653577
0.2	0.5	1.239377	0.653181
0.2	0.6	1.239406	0.652992
0.2	0.7	1.239447	0.652956
0.2	0.8	1.240057	0.653665
0.3	0	1.256441	0.672366
0.3	0.1	1.256441	0.672366
0.3	0.2	1.256441	0.672366
0.3	0.3	1.255547	0.671504
0.3	0.4	1.255646	0.671539
0.3	0.5	1.254945	0.671027
0.3	0.6	1.25367	0.67016
0.3	0.7	1.253555	0.670078
0.3	0.8	1.253555	0.670078
0.4	0	1.237954	0.651068
0.4	0.1	1.237954	0.651068
0.4	0.2	1.237954	0.651068
...	...	...	...
0.6	0.6	1.240557	0.654569
0.6	0.7	1.240557	0.654569
0.6	0.8	1.240557	0.654569
0.7	0	1.243678	0.658551
0.7	0.1	1.243678	0.658551
0.7	0.2	1.243678	0.658551
0.7	0.3	1.243678	0.658551
0.7	0.4	1.243678	0.658551
0.7	0.5	1.243678	0.658551

Table 5.6 Tuning colsample\_bytree and colsample\_bylevel

colsample_bytree	colsample_bylevel	rmse	mape_pct
0.5	0.5	1.308963	0.714537
0.5	0.6	1.308963	0.714537
0.5	0.7	1.308963	0.714537
0.5	0.8	1.317871	0.720343
0.5	0.9	1.317871	0.720343
0.5	1	1.291837	0.702152
0.6	0.5	1.308963	0.714537
0.6	0.6	1.308963	0.714537
0.6	0.7	1.308963	0.714537
0.6	0.8	1.317871	0.720343
0.6	0.9	1.317871	0.720343
0.6	1	1.291837	0.702152
0.7	0.5	1.258588	0.675274
0.7	0.6	1.258725	0.673768
0.7	0.7	1.258725	0.673768
0.7	0.8	1.267979	0.682271
0.7	0.9	1.267979	0.682271
0.7	1	1.244929	0.660852
0.8	0.5	1.363271	0.75974
0.8	0.6	1.363271	0.75974
0.8	0.7	1.27305	0.681167
0.8	0.8	1.27305	0.681167
0.8	0.9	1.249358	0.661877
0.8	1	1.245122	0.661033
0.9	0.5	1.391633	0.77633
0.9	0.6	1.304644	0.70756
0.9	0.7	1.304644	0.70756
0.9	0.8	1.277105	0.687764
0.9	0.9	1.285495	0.695538
0.9	1	1.252295	0.667862
1	0.5	1.302522	0.708703
1	0.6	1.302522	0.708703
1	0.7	1.249387	0.664993
1	0.8	1.250705	0.666865
1	0.9	1.252033	0.66838
1	1	1.237304	0.650484



0.7	0.6	1.243678	0.658551
0.7	0.7	1.243678	0.658551
0.7	0.8	1.243678	0.658551
0.8	0	1.253708	0.670101
0.8	0.1	1.253708	0.670101
0.8	0.2	1.253708	0.670101
0.8	0.3	1.253708	0.670101
0.8	0.4	1.253708	0.670101
0.8	0.5	1.253708	0.670101
0.8	0.6	1.253708	0.670101
0.8	0.7	1.253708	0.670101
0.8	0.8	1.253708	0.670101
1	0	1.239485	0.653097
1	0.1	1.239485	0.653097
1	0.2	1.239485	0.653097
1	0.3	1.239485	0.653097
1	0.4	1.239485	0.653097
1	0.5	1.239485	0.653097
1	0.6	1.239485	0.653097
1	0.7	1.239485	0.653097
1	0.8	1.239485	0.653097

Table 18 Tuning subsample and gamma parameter; Tuning colsample\_bytree and colsample\_bylevel

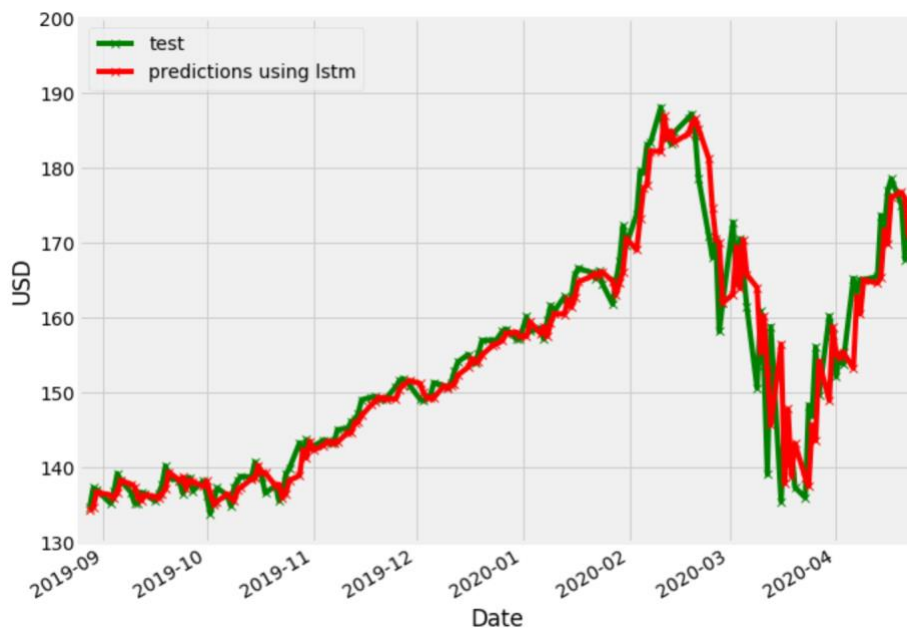


Figure 24 MSFT stock price forecast using LSTM

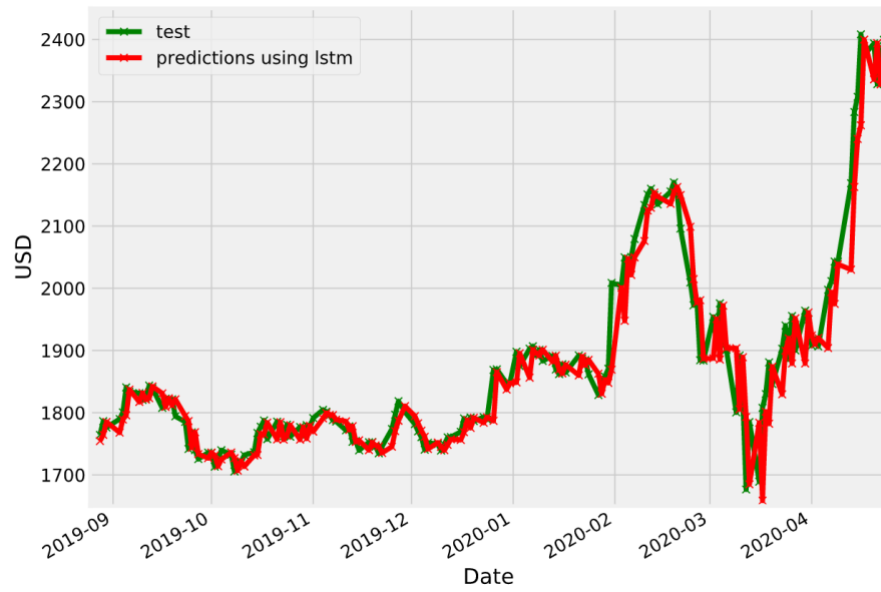


Figure 25 AMZN stock price forecast using LSTM

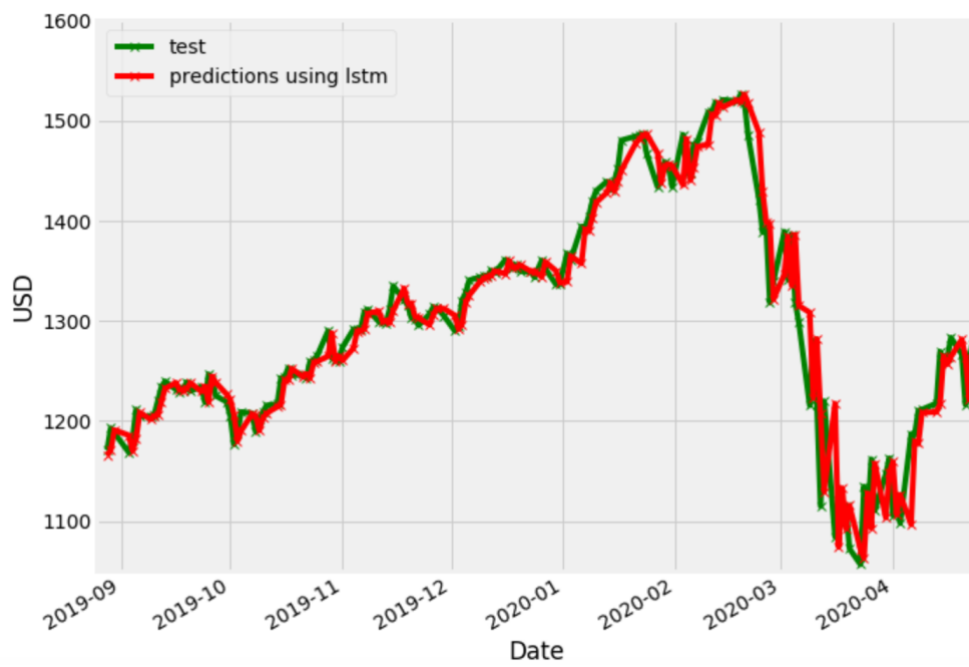


Figure 26 GOOG stock price forecast using LSTM

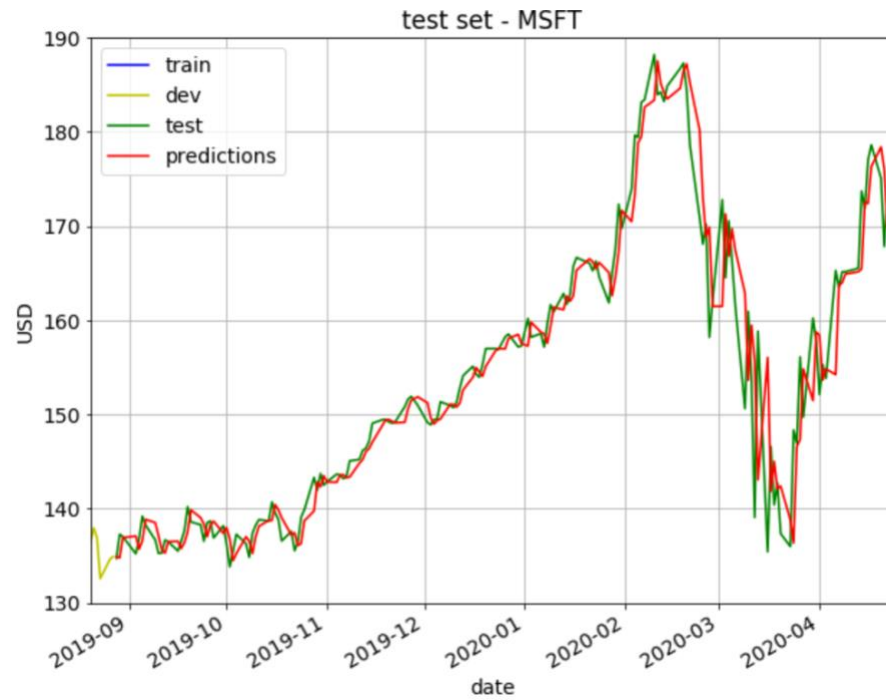


Figure 27 MSFT stock price forecast using XGBoost

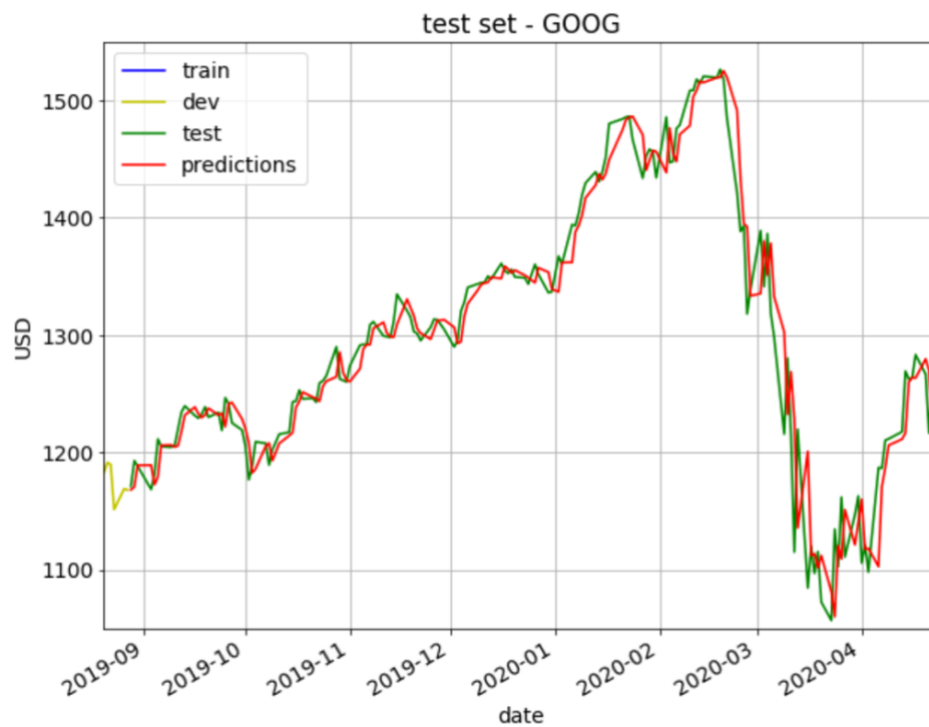


Figure 28 GOOG stock price forecast using XGBoost

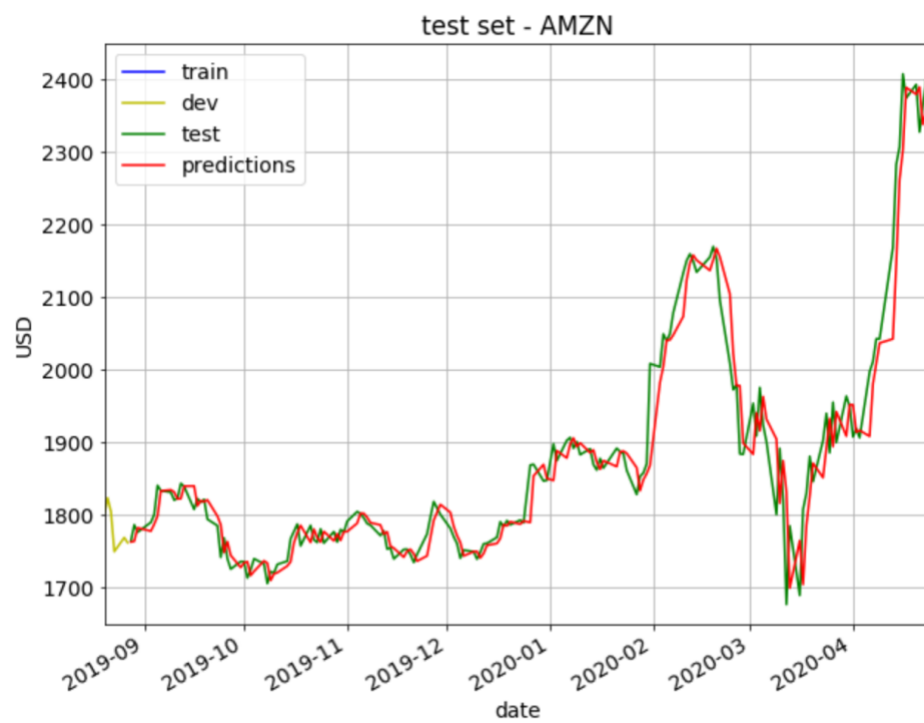


Figure 29 AMZN stock price forecast using XGBoost

## References

- [1] W. McKinney, J. Perktold, S. Seabold “ Time Series Analysis in Python with statsmodels”, Page 107-113, Proc. of the 10th python in science conf. (scipy 2011)
- [2] Wikipedia, “Partial autocorrelation function”,  
[https://en.wikipedia.org/wiki/Partial\\_autocorrelation\\_function](https://en.wikipedia.org/wiki/Partial_autocorrelation_function)
- [3] SARIMAX: Introduction,  
[https://www.statsmodels.org/dev/examples/notebooks/generated/statespace\\_sarimax\\_stata.html](https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html)
- [4] Wikipedia, Ljung-Box Test, [https://en.wikipedia.org/wiki/Ljung%E2%80%93Box\\_test](https://en.wikipedia.org/wiki/Ljung%E2%80%93Box_test)
- [5] Understanding LSTM Networks, <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [6] What is "units" in LSTM layer of Keras? <https://zhuanlan.zhihu.com/p/58854907>
- [7] Adam documentation, <https://keras.io/api/optimizers/adam/>
- [8] Jerome H. Friedman, “Greedy Function Approximation: A Gradient Boosting Machine”, The Annals of Statistics 2001, Vol. 29, No. 5, 1189–1232
- [9] Tianqi Chen, Carlos Guestrin, “XGBoost: A Scalable Tree Boosting System”, KDD ’16, August 13-17, 2016, San Francisco, CA, USA
- [10] XGBoost documentation, <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>
- [11] Jason Brownlee, A Gentle Introduction to XGBoost for Applied Machine Learning,  
<https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>
- [12] Hastie, T.; Tibshirani, R.; Friedman, J. H. (2009), "10. Boosting and Additive Trees", The Elements of Statistical Learning (2nd ed.). New York: Springer. pp. 337–384.
- [13] Boosting algorithm: XGBoost, <https://towardsdatascience.com/boosting-algorithm-xgboost-4d9ec0207d>

- [14] Thomas Vincent, “A Guide to Time Series Forecasting with ARIMA in Python 3”,  
<https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arma-in-python-3>
- [15] Campbell, John Y. 2014. “Empirical Asset Pricing: Eugene Fama, Lars Peter Hansen, and Robert Shiller.” Working Paper, Department of Economics, Harvard University.  
<http://nrs.harvard.edu/urn-3:HUL.InstRepos:25142544>
- [15] Akaike information criterion, [https://en.wikipedia.org/wiki/Akaike\\_information\\_criterion](https://en.wikipedia.org/wiki/Akaike_information_criterion)
- [16] Autocorrelation and partial autocorrelation plot; <https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/time-series/how-to/arma/interpret-the-results/all-statistics-and-graphs/graphs/#pacf-of-residuals>
- [17] Datacamp, Chapter four Seasonal time series Arima models in python
- [18] Wikipedia, Mean Absolute Percentage Error,  
[https://en.wikipedia.org/wiki/Mean\\_absolute\\_percentage\\_error](https://en.wikipedia.org/wiki/Mean_absolute_percentage_error)
- [19] Jason Brownie, How to Create an ARIMA Model for Time Series Forecasting in Python,  
<https://machinelearningmastery.com/arma-for-time-series-forecasting-with-python/>
- [20] Jason Brownie, How to grid search ARIMA model hyperparameters with Python,  
<https://machinelearningmastery.com/grid-search-arma-hyperparameters-with-python/>